INFERENCE WITH CLASSIFIERS: A STUDY OF STRUCTURED OUTPUT PROBLEMS IN NATURAL LANGUAGE PROCESSING

BY

VASIN PUNYAKANOK

B.Eng., King Mongkut's Institute of Technology Ladkrabang, 1995
M.S., University of Illinois at Urbana-Champaign, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

Abstract

A large number of problems in natural language processing (NLP) involve outputs with complex structure. Conceptually in such problems, the task is to assign values to multiple variables which represent the outputs of several interdependent components. A natural approach to this task is to formulate it as a two-stage process. In the first stage, the variables are assigned initial values using machine learning based programs. In the second, an inference procedure uses the outcomes of the first stage classifiers along with domain specific constraints in order to infer a globally consistent final prediction.

This dissertation introduces a framework, *inference with classifiers*, to study such problems. The framework is applied to two important and fundamental NLP problems that involve complex structured outputs, shallow parsing and semantic role labeling. In shallow parsing, the goal is to identify syntactic phrases in sentences, which has been found useful in a variety of large-scale NLP applications. Semantic role labeling is the task of identifying predicate-argument structure in sentences, a crucial step toward a deeper understanding of natural language. In both tasks, we develop state-of-the-art systems which have been used in practice.

In this framework, we have shown the significance of incorporating constraints into the inference stage as a way to correct and improve the decisions of the stand alone classifiers. Although it is clear that incorporating constraints into inference necessarily improves global coherency, there is no guarantee of the improvement in the performance measured in terms of the accuracy of the local predictions—the metric that is of interest for most applications. We develop a better theoretic understanding of this issue. Under a reasonable assumption, we prove a sufficient condition to guarantee that using constraints cannot degrade the performance with respect to Hamming loss. In

addition, we provide an experimental study suggesting that constraints can improve performance even when the sufficient conditions are not fully satisfied.

In memory of Grandma

Acknowledgments

Being able to complete this dissertation is very special for me, but far more special than that is every wonderful person who helped compose this journey.

First, I am greatly in debt to my advisor, Dan Roth, without whom my journey would have been lost. His unending advice and support has been the light that showed me not only the path that I intended to walk, but also many new ways that I never thought of. I am also admired his patience for that he has never shown even a bit of disappointment no matter how slow and clumsy I am, at times, as a student and a researcher. Besides his advice and support as an excellent advisor, I am always amazed by his balance of career and life. On another day of procrastination, I went to a tennis match only to see him finish his long day of work to cheer up his son being a ballboy on the court. In addition to his career achievement, I wish I could one day build such a lovely family.

Most of my time during this graduate study was spent with the Cognitive Computation Group. I enjoyed the privilege of working with all of my fellow group members as well as our visitors: Fabio Aiolli, Shivani Agarwal, Rodrigo de Salvo Braz, Andy Carlson, Xavier Carreras, Chad Cumby, Yair Even-Zohar, Roxana Girju, Peter Koomen, Yuval Krymolowski, Xin Li, Lluís Màrquez, Paul Morie, Marcia Muñoz, Ramya Nagarajan, Nick Rizzolo, Jeff Rosen, Mark Sammons, Kevin Small, Yuancheng Tu, Wen-tau Yih, Dmitry Zelenko, and Dav Zimak. I am very grateful to all of them for their help, and despite whether we have ever worked directly in the same projects, the butterfly toy we threw across the office cubicles has surely made this journey entertaining.

My special appreciation is to Scott Wen-tau Yih and Dav Zimak. They have touched every part of the work in this dissertation as well as this journey. Their ideas, labor, suggestions and comments are a tremendous part of this work, and their friendship has made this journey a very

enjoyable one. With all my heart, I really wish them well and successful in every aspect of their lives. I owe you guys a big time, and if you read this, please let me know how I can repay you.

I would also like to thank my dissertation committee, Dan Roth, Gerald DeJong, Steven LaValle, and ChengXiang Zhai for their comments and suggestions not only on this dissertation but also in a broader perspective. Their words will certainly prove invaluable to my future.

And, finally, larger than my life is always the everlasting love and support from my family. I am sorry to Grandma that I could not make this happen in time. Her soft touch will be remembered forever.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The traditional classification problem in machine learning is the problem of identifying a class label for any given input according to some function that the machine has previously learned from training examples. Specifically, *binary classification* is a task that distinguishes examples into one of the two classes, e.g. a medical testing to identify cancerous patients. Progress has been made within this area of research resulting in the developments of learning algorithms, such as Perceptron (Rosenblatt, 1957), Winnow (Littlestone, 1988), and support vector machine (Cortes and Vapnik, 1995), which have been successfully applied to many real world applications. In addition, the theoretical understanding of the hardness of the problem, and the generalization of learning algorithms has been well studied.

More generally, the problem of *multiclass classification* is the problem in which the number of labels grows beyond two, e.g., the problem of handwritten digit recognition. Early methods include the extension of binary learners to cope with multiclass problems by reducing them to many binary classification problems. For example, the one-versus-all technique formulates an l-class problem as l binary problems where the goal of each learner is to distinguish the examples in a class versus those in the rest. Another line of research tackles the multiclass classification problem directly and overcomes the weakness of those techniques that rely on reformulating the problem into multiple independent binary classification (Har-Peled, Roth, and Zimak, 2003). Theoretical understanding within this area has also been well studied.

Although multiclass classification allows one to model complex some classification problems, a large number of problems, especially in natural language processing, involve outputs with a more complex structure resulting in very large number of possible outcomes. This makes the

direct application of multiclass learning practically infeasible. Moreover, by viewing each possible assignment to the output variables as an individual class label, direct multiclass classifications do not exploit the structure and the dependencies of the output variables—information that can simplify the learning task, and may be vital to successful learning.

Consider, for example, the problem of extracting entities and relation in a sentence (Roth and Yih, 2004). In this task, the goal is to recognize the relations, such as *kill*, and *live in*, in the sentence. At the same time, the recognizer needs to identify and classify the entities that involve in the relation into different types, i.e. *person*, organization, and *location*. For example, given the following sentence, "`J. V. Oswald was murdered at JFK, and his assassin, R. U. Washington...`", the goal is to identify the relation *kill (R. U. Washington*$_{person}$*, J. V. Oswald*$_{person}$*)*. In this case, the potential outputs includes all possible pairs of entities which can be any substrings in the sentences, and the total number of labels becomes very large.

Instead of modeling *structured output problems* as multiclass classification problems, a natural approach is to make *local* decisions, and, then, to combine them into a *global* outcome by an inference procedure. For example in relation-entity recognition, one may split the problem into, first, identifying entities and, then, the relations. The identification of entities involves predicting for each substrings in the sentence whether or not it is an entity, and what type. Then the relation recognizer decides the relations between each entity. Clearly, this introduces constraints among the outputs. For example, entities do not overlap, that is, for the above example, if *J. V. Oswald* is identified as an entity, *J. V.* alone must not be. Additionally, each relation cannot take all possible types of entities as its arguments. For example, a *location* cannot kill a *person*. Therefore, the procedure that is used to infer the final predictions must be able to handle these constraints.

For a more concrete example, consider, the problem of *chunking* natural language sentences where the goal is to identify several kinds of phrases (e.g. noun phrases and verb phrases) in sentences. Instead of directly predicting for each possible chunk in the sentence whether it is a phrase or not, the most common approach is to represent the final output by a sequence of symbols in some specific representation, and makes prediction for each symbol in the sequence (Ramshaw

and Marcus, 1995; Muñoz et al., 1999; Tjong Kim Sang and Buchholz, 2000; Punyakanok and Roth, 2001). An example is to represent an output by the sequence of symbols that indicate the beginnings and the ends of chunks. Given such representation, one way to address the problem is to use two classifiers for each type of phrases, one of which recognizes the beginning of the phrase, and the other its end. By modeling the problem as several sub-problems, learning becomes simpler. Then, to combine the outcomes of these classifiers, the inference procedure must maintain the alternative order of beginnings and ends in order to produce the final legitimate non-overlapping phrases.

The previous examples illustrate an approach that has been extensively used to solve complex problems that is to separate the problems to two levels. In the first, several (learned) classifiers output a classification along with a confidence (or a conditional probability). The second is the inference level—a process which takes into account the outcomes of the first level along with additional domain or problem specific constraints to infer the final outcome for the global problem in a way that is coherent with the classifiers and respects the constraints.

It is important to note here that separating the problems to two levels does not necessary dictate how classifiers are trained. Although, the natural approach is to train the classifiers independently of the inference, and use the inference only at the evaluation time, approaches for training classifiers together with the inference have also been studied recently. Incorporating the inference directly into learning can lead to solutions that directly optimize the true global objective function, and, therefore, improve overall performance. Nonetheless, there has not been much evidence in real world applications to support this point. Although, this approach is very fundamentally attractive, its lack of conceptual simplicity and efficiency has made it less popular than the approach that decouples training from inference. Moreover, it requires the availability of the training examples for all local problems that are present in the same context of the global problem which can be very expensive to obtain. In addition, as the complexity of real world problems grows, it is less likely that the coupling approach will scale up well. On the other hand, because of its modularity, the decoupling the training from the inference is conceptually simpler and more efficient. In addition,

it allows incorporating previously designed and learned classifiers which is more likely to be a key benefit especially when real world problems become larger and involve integrations of smaller components that have been previously developed.

The other benefit of separating inference from learning is in the ease of incorporating domain specific knowledge in terms of constraints for the inference procedure. Often, the constraints are not present at the time of developing each component, but different constraints become available only when the components are used for different tasks. For example, in relation-entity recognition, at first, the entity recognizer may be trained unaware that relations exist. Only later when the concept of relation becomes available are the constraints over the compatibility of entity types and relations introduced. Moreover, the same entity recognizer may be used in other tasks with different types of constraints. Consider another example, name tracing problem (Li, Morie, and Roth, 2005) where the goal is to identify whether different occurrences of entities in document refer to the same entities. Obviously, if two occurrences of entities refer to the same entity, they should be recognized as the same type—this can be posed as a constraint to the output of entity recognizers for this specific task.

Intuitively, incorporating constraints at the inference stage would result in more coherent output—one in which the values assigned to different components do not contradict one another. Consider again the example, "`J. V. Oswald was murdered at JFK, and his assassin, R. U. Washington...`," in the relation-entity recognition task. If the entity *R. U. Washington* is recognized as a *location* while the relation *kill(R. U. Washington, J. V. Oswald)* is detected, the constraints that a *location* cannot kill a *person* may help correcting the mistakes by the entity recognizer. However, the question of whether incorporating constraints actually improves the measured performance of the system depends on how the performance is evaluated. For example, if correct outputs are accepted only if both relations and entities are recognized correctly, the constraint as shown above guarantees to fix only bad outputs, and hence, can only improve the performance. On the other hand, if the performance is measured in terms of some combination of the performance of the individual classifiers such as Hamming distance that considers the overall

mistakes as the accumulation of the mistakes by individual classifiers, instead of changing the entity type of *R. U. Washington* from a *location* to a *person*, the use of the constraint may incorrectly fixes the relation *kill(R. U. Washington, J. V. Oswald)* to *live_in(J. V. Oswald, R. U. Washington)* resulting in more mistakes.

Despite the fact that constraints have been consistently shown to improve the performance in practice (Roth and Yih, 2005), there is no theoretical guarantee that the additional constraints would help since the classifiers were trained without the knowledge of the extra constraints. Indeed, it is possible to show that constraints can degrade the performance (see Chapter 6). Unless constraints are necessary for a problem of which outputs must satisfy, additional problem-specific knowledge that are encoded in terms of constraints at the inference procedure should be used with care or it can degrade the overall performance. Theoretical understanding on the issue of the usefulness of constraints in an inference procedure would help explain when and why constraints can be safely used which may lead to a better inference procedure or a better learning algorithm that has its goal to be incorporated in this framework. This issue, however, has not received much attention.

The focus of this dissertation is to the study of learning problems with complex and structured output. The dissertation we develop throughout this document is that of *divide and conquer*; we suggest that decomposing the output into components, learning each one, and then using these, along with task and domain specific constraints to infer a coherent output, is a good approach to take for these tasks. We call such approach as *Inference with Classifiers*. Formally, we formulate how complex problems may be viewed in two levels, learning and inference. Our study mostly focuses on the inference level. First, we show the usefulness of the constraints in real world applications by applying the formulation to two important fundamental problems in natural language processing—shallow parsing (Abney, 1991; Grefenstette, 1993; Harris, 1957) and semantic role labeling (Kingsbury and Palmer, 2002). In addition, we aim to explain the relations of the constraints in the inference to the global performance of a system.

1.1 Contributions

The study results in the three main contributions. First, we rigorously formulate the problem of inference with classifiers which have been used extensively in practice but has never been viewed in a unified framework. Formulating the problem rigorously allows us to ask the questions of the usefulness of the constraints in a formal way for which we are able to partially answer, and hopefully, will bring about new research direction in the machine learning community.

Second, we develop one of state-of-the-art systems in the task of shallow parsing. Shallow parsing problem is the task to identify some syntactical phrases in sentences, and has been found useful in a varieties of large-scale natural language processing applications including information extraction, text summarization, and question answering systems (Appelt et al., 1993; Grishman, 1995; Roth et al., 2001). In addition, our approaches here may be directly applied to different problems of the similar nature—identifying some patterns in sequences—in any domain. One of our approaches has been successfully applied to an application in the computational biology (Chuang and Roth, 2001).

Finally, our study of inference with classifiers on the semantic role labeling task results in the state-of-art system which was the top system among those participated in the shared task organized by the 2005 Computational Natural Language Learning Conference (CoNLL05) (Carreras and Màrquez, 2005). Semantic role labeling advances beyond syntactical analysis of natural language sentences, by identifying, for each predicate (verb) in a sentence, its semantic roles such as such as Agent, Patient or Instrument, and their adjuncts, such as Locative, Temporal or Manner. This is an important task toward a deeper understanding of natural language, and has already been used for such task as deciding whether the meaning of one sentence entails that of the other (textual entailment task) (Braz et al., 2005). The approach developed here can also be flexibly extended to different problems, not limited to sequential problems.

1.2 Outline

The rest of the dissertation is organized as follows. Chapter 2 introduce the structured output problems in natural language processing and presents some background materials that will be used in this dissertation. Specifically, we describe two machine learning approaches, i.e. naïve Bayes classifiers and SNoW learning architecture which are used in the experiments in this dissertation. In addition, we present three specific inference frameworks, i.e. hidden Markov models (HMM), constraint-satisfaction problem (CSP), and integer linear programming (ILP) which will be the fundamental tools to develop the inference procedures in later chapters.

Chapter 3 formally introduces the *inference with classifiers* framework. In addition, the chapter discusses how different inference methods can be used to combine the outcomes of classifiers. A short survey on how learning can be done in different scenarios is also presented.

Chapter 4 presents three inference with classifiers based approaches to address the shallow parsing problem which is the task to identify syntactic phrases in natural langauge sentences. In the first two approaches, the inference procedures are based on a probabilistic model within a Markovian framework. The other approach extends a constraint satisfaction formalism to deal with variables that are associated with costs which can be used to model the problem of inference with classifiers. All three approaches are evaluated experimentally on the tasks of identifying base noun phrases and subject-verb patterns.

Chapter 5 discusses how the inference with classifiers approach is applied to the problem of semantic role labeling (SRL) which is the task to identify the arguments of each verb-predicate in a sentence. Specifically, we develop two systems. Both decouple learning from the inference which is formulated as an integer linear program. The first system uses only partial parsing information, and is empirically evaluated on the CoNLL-2004 shared task dataset. The second system makes use of full parsing information with an extension of the inference procedure to combine the outputs from multiple SRL systems. The system is evaluated on the CoNLL-2005 shared task dataset, and it is the best system in the competition.

Chapter 6 investigates the influence of constraints in inference with respect to two specific loss functions. We show that, with respect to zero-one loss, the performance of a system can be shown not to degrade with the use of constraints while this is not true for Hamming loss. In case of Hamming loss, we develop a sufficient condition to guarantee that using constraints cannot hurt the system performance. In addition, the chapter presents an empirical study on a real world system which leads to some explanations why constraints do not impair the system even if the sufficient condition is not fully satisfied.

Finally, Chapter 7 concludes this dissertation and discusses several directions for the future research.

1.3 Publication Notes

This dissertation includes partially the materials that appear in the following articles.

- P. Koomen, V. Punyakanok, D. Roth, and W. Yih. 2005. Generalized Inference with Multiple Semantic Role Labeling Systems. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 181–184.

- V. Punyakanok, D. Roth, and W. Yih. 2005. The Necessity of Syntactic Parsing for Semantic Role Labeling. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1117–1123.

- V. Punyakanok, D. Roth, W. Yih, and D. Zimak. 2004. Semantic Role Labeling via Integer Linear Programming Inference. In *Proceedings the 20th International Conference on Computational Linguistics (COLING)*, pages 1346–1352.

- V. Punyakanok, D. Roth, W. Yih, D. Zimak, and Y. Tu. 2004. Generalized Inference with Multiple Semantic Role Labeling Systems. In *Proceedings of CoNLL-2004*, pages 130–133.

- V. Punyakanok and D. Roth. 2001. The Use of Classifiers in Sequential Inference. In *Advances in Neural Information Processing Systems 13 (NIPS)*, pages 995–1001.

- M. Muñoz, V. Punyakanok, D. Roth, and D. Zimak. 1999. A Learning Approach to Shallow Parsing. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP-VLC)*, pages 168–178.

Chapter 2

Background

This chapter introduces the structured output problems in natural language processing which are the main applications in this dissertation.

The chapter is organized in three sections. First, we introduce the traditional classification problem, and specifically, two machine learning approaches, i.e. naïve Bayes classifiers and SNoW learning architecture, which will be used in the experiments in this dissertation. Next, we introduce the structured output problems and give an overview of the approaches taken to tackle the problems which naturally leads to the issue of inference. In the last section, three specific inference frameworks, i.e. hidden Markov models (HMM), constraint-satisfaction problem (CSP), and integer linear programming (ILP) are introduced. These three frameworks are the fundamental tools that we use to develop the inference procedures in later chapters.

2.1 Classification Problems

The traditional classification problem in machine learning is the problem of identifying a class label for any given input according to some function that the machine has previously learned from training examples. Formally, the goal is to learn a function (*hypothesis*) $f : \mathcal{X} \rightarrow \mathcal{Y}$ mapping from an input $\mathbf{x} \in \mathcal{X}$ in some domain to a single label $y \in \mathcal{Y}$ in some finite space given a set of training examples $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$. If $|\mathcal{Y}| = 2$, the problem is *binary classification*, and *multiclass classification* is that with $|\mathcal{Y}| > 2$.

Given an example $(\mathbf{x}, y)$, the quality of the output of a hypothesis f on $\mathbf{x}$, i.e. $f(\mathbf{x})$, is measured in terms of some loss function $\ell(y, f(\mathbf{x}))$. For example, the quality is commonly measured in terms

of misclassification:

$$\ell(y, y') = \begin{cases} 0, & \text{if } y = y', \\ 1, & \text{if } y \neq y'. \end{cases}$$

The overall quality of a hypothesis f is then measured in terms of expected loss over all examples:

$$\ell(f) = \mathbf{E}_{\mathbf{x},y}\ell(y, h(\mathbf{x})) = \sum_{\mathbf{x}\in\mathcal{X}, y\in\mathcal{Y}} P(\mathbf{x}, y)\ell(y, h(\mathbf{x})),$$

where $\mathbf{E}_{\mathbf{x},\mathbf{y}}$ denotes the expectation with respect to the true underlying (possibly unknown) distribution of $(\mathbf{x}, \mathbf{y})$.

Hence, the optimal classifier that minimizes the loss is:

$$f^*(\mathbf{x}) = \arg\min_{y\in\mathcal{Y}} \sum_{\mathbf{x}\in\mathcal{X}, y\in\mathcal{Y}} P(y|\mathbf{x})\ell(y, h(\mathbf{x})).$$

This is known as *Bayes optimal classifier*.

In practice, however, the underlying probability distribution is unknown. Different learning algorithms, therefore, have been invented and analyzed in order to approximate the true hypotheses based on the training data. We introduce here only two learning approaches that will be used in the experiments in this dissertation, i.e. naïve Bayes classifiers, and SNoW learning architecture.

2.1.1 Naïve Bayes Classifiers

Assume that each input $\mathbf{x}$ contains a list of attributes $\langle x_1, x_2, \ldots, x_n\rangle$. A naïve Bayes classifier is a probabilistic classifier that assumes independence between attributes given that the underlying output class is known. Formally, this assumption can be written as

$$P(x_1, x_2, \ldots, x_n|y) = \prod_i^n P(x_i|y).$$

Hence, in predicting an output, naïve Bayes classifiers choose the label that maximizes poste-

rior probability computed by

$$\begin{aligned} y &= \arg\max_{y \in \mathcal{Y}} P(y|x_1, x_2, \ldots, x_n) \\ &= \arg\max_{y \in \mathcal{Y}} \frac{P(x_1, x_2, \ldots, x_n|y)P(y)}{P(x_1, x_2, \ldots, x_n)} \\ &= \arg\max_{y \in \mathcal{Y}} \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, x_2, \ldots, x_n)} \\ &= \arg\max_{y \in \mathcal{Y}} P(y)\prod_{i=1}^{n} P(x_i|y). \end{aligned}$$

$P(x_1, x_2, \ldots, x_n)$ can be dropped because the goal is to find the optimal output, not the actual probability.

Training a naïve Bayes classifier is usually done by maximizing the likelihood of the training data:

$$\begin{aligned} L(\mathcal{D}|P) &= \prod_{(\mathbf{x},y)\in\mathcal{D}} P(\mathbf{x}, y) \\ &= \prod_{(\mathbf{x},y)\in\mathcal{D}} \prod_{i=1}^{n} P(y)P(x_i|y), \end{aligned}$$

where $\mathcal{D}$ is the training dataset. That is each term in the product can be computed by directly taking statistics from the training dataset. However, in many applications such as in natural language processing where the sparsity of the data is a common problem, simple maximum likelihood estimation leads to zero probability of unseen attributes, and hence an undefined prediction. To overcome this, different *smoothing* techniques depending on the applications, e.g. adding a fixed small constant to the frequency count (Mitchell, 1997), may be used to prevent this zero probability events to dominate the computation.

Despite the simplicity of naïve Bayes classifiers, it can be shown that the hypothesis space of naïve Bayes is equivalent to that of linear classifiers (Jaeger, 2003). More surprisingly, even with the independence assumption which almost always never hold in real world applications, naïve

Bayes, many times, performs very well in practice (Domingos and Pazzani, 1997; Elkan, 1997). Some theoretical explanation of this phenomenon may be found in Roth (1999) and Garg and Roth (2001).

2.1.2 Sparse Network of Winnow (SNoW)

SNoW (Sparse Networks of Winnows) (Carlson et al., 1999; Roth, 1998) is a multiclass classifier that employs many linear functions to represent its hypothesis. In its most basic form, SNoW architecture consists of two layers. The first layer is the input layer which contains a collection of nodes representing different features. The second layer contains many target nodes each of which represents a different concept class. Each target node is linear function $\sum_{i \in \mathcal{A}_t} w_{t,i} s_i$ where $\mathcal{A}_t$ is the set of the feature indices that participate in the concept class t, $w_{t,i}$ is the weight of the feature i in class t, and s_i represents the activity of feature i which can be some Boolean value representing whether or not the feature is active, or some real value representing its strength.

SNoW implements various learning algorithms including Perceptron (Rosenblatt, 1957), naïve Bayes and Winnow (Littlestone, 1988). Naïve Bayes as described in Section 2.1.1 is a batch learning algorithm which observe all training examples at once and infers its hypothesis based on the statistics of the examples as a whole. On the other hand, Perceptron and Winnow are online and mistake-driven learning algorithms which process one example at a time and update their hypotheses (weight vectors) only when a mistake occurs. Winnow differs from Perceptron in its update rule which is done in a multiplicative fashion rather than using an additive update. This key difference results in the number of examples required by Winnow to learn a linear function growing linearly with the number of relevant features and only logarithmically with the number of all features. This property along with the design of SNoW to deal with sparse features makes SNoW extremely suitable for large scale learning tasks, especially those in natural language processing that involves very large feature space, but only a relatively few features are active in each example.

Various extensions are implemented in SNoW to enhance its generalization. Such extensions include regularization which is implemented in SNoW in the form of *thick hyperplane* (Dagan,

Karov, and Roth, 1997; Grove and Roth, 2001; Li et al., 2002; Zhang, Damerau, and Johnson, 2002). *Voted Perceptron* (Freund and Schapire, 1999) is another extension that allows SNoW to make predictions by conceptually taking the votes from all hypotheses obtained during the training time instead of using only the final one.

Typically, SNoW is used as a classifier, and makes predictions using a winner-take-all mechanism over the raw activation values (the linear summations) of the target classes. However, when the score of each class is required, such score is computed using softmax (Bishop, 1995) over the raw activation value, namely, the exponential function of the output, normalized across all classes to sum to 1:

$$f(i) = \frac{e^{act_i}}{\sum_{1 \leq j \leq n} e^{act_j}},$$

where act_i is the raw activation value of class i. It can be shown empirically that the activation levels of SNoW roughly correspond to the posterior probabilities of the targets given the input (Rosen, 1999).

SNoW has been successfully applied to many large scale learning problems in natural langauge processing (Golding and Roth, 1999; Muñoz et al., 1999; Punyakanok and Roth, 2001; Roth, 1998; Roth and Zelenko, 1998; Shen and Joshi, 2003), bioinformatics (Chuang and Roth, 2001), and visual processing (Agarwal and Roth, 2002; Roth, Yang, and Ahuja, 2002). The complete details of SNoW can be found in its manual (Carlson et al., 2004).

2.2 Structured Output Problems

A large number of problems, especially in natural language processing, involve outputs with complex structure. A well illustrative example is the problem of syntactical parsing in natural language which is the problem of finding the syntactical structure, i.e. parse tree, of a sentence.

Traditionally, the structured output problems are tackled by assuming some underlying probabilistic generative model that respects the structure of the problem. A probabilistic generative model is a model that specifies the probability distribution of data, which may contain both ob-

served and unobserved variables, by explicitly describing the process that generates the data. That is, the model defines the joint probability distribution $P(\mathbf{x}, \mathbf{y})$ where $\mathbf{x}$ is the input to the problem, and $\mathbf{y}$ is the output. Note that unlike the traditional classification problem, here the output is usually a collection of many variables instead of a single one. In syntactical parsing, current state-of-the-art parsers (Collins, 1999; Charniak, 2001) are generative models that describe how parse trees (outputs) and their corresponding sentences (inputs) are generated. Given such model, the goal is to find the most likely parse tree for any given input sentence:

$$\begin{aligned} \mathbf{y} &= \arg\max_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y}|\mathbf{x}) \\ &= \arg\max_{\mathbf{y} \in \mathcal{Y}} \frac{P(\mathbf{y}, \mathbf{x})}{P(\mathbf{x})} \\ &= \arg\max_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y}, \mathbf{x}). \end{aligned}$$

Generally, a generative model is intuitive to model different structures; however, it offers low discriminative power because the training of the model is usually done to maximize the likelihood of the training data, not to optimize the true objective of the problem, e.g. to maximize the accuracy of the outputs given the inputs. Although, there have been some attempts, e.g. Kakade, Teh, and Roweis (2002), to improve this by developing different criteria for training the generative models which better capture the true objective functions, the need to specify the underlying process beforehand still makes it hard to incorporate new knowledge into an already existing model.

The alternative of a generative model is a discriminative model which makes no attempt to define an underlying process that generates the inputs. Instead the attempt is to create a function that directly predicts the outputs given the inputs. In probabilistic framework, such model is sometimes called *conditional model*, e.g. maximum entropy Markov models (MEMM) by McCallum, Freitag, and Pereira (2000), because it directly specifies the conditional probability $P(\mathbf{y}|\mathbf{x})$. The training of such model is done with the objective that more directly corresponds to the true objective of the problem. In non-probabilistic framework, the approach that is taken along this direction is to decompose the model into many smaller sub-models and locally train each sub-model inde-

pendently using standard machine learning classification techniques. An example in syntactical parsing is the maximum entropy parser by Ratnaparkhi (1996).

Recently, there has been some progress in developing global approaches that do not explicitly decompose the model for localized discriminative training. Instead, the whole model is trained together. A good sample of such models include discriminative training for HMMs (Collins, 2002), conditional random fields (Lafferty, McCallum, and Pereira, 2001), and max-margin Markov networks (Taskar, Guestrin, and Koller, 2004) which we will discuss in more details in Section 3.3.1. The benefit of the global training is the model that is trained to optimize the true global objective function; however, the common downside is the increase in the complexity of the training procedure. In addition, the benefit of the global training approaches in real world applications are not significantly observed, and hence, the relative merits between these two approaches are still an open question.

In any case, one can view structured output problems as those that involve multiple output variables that interact in some ways which are dictated by the structure of the problem. For example, one can view the syntactic parsing problem as a problem that contains many variables, each of which represents a possible constituent in the parse tree. The goal is to decide whether each of this constituent actually appears in the output parse tree, and if so, what the label is. In order to output a legitimate parse tree, the final output must respect the tree structure, i.e. the output constituents may embed one another but cannot overlap. In this view, it is not hard to see that a structured output problem may be viewed as a problem of inference with classifiers that consists of two levels. In the first, several (learned) classifiers are evaluated and output classifications along with their confidence values (or conditional probabilities). The second is the inference level—a process which takes into account the outcomes of the different classifiers along with some domain specific constraints, and infers the final outcome for the global problem in a way that is coherent with the constraints.

Looking at the problem as a problem of inference of classifiers allows one to visualize easily the generalization of the structured output problems to many other well-known problems. A simple

exercise is the multiclass classification that may be viewed as containing many binary variables, one per class. The goal is to decide for each class whether or not the given input belong to that class with the constraint that the input can belong to only one class.

Solving multiclass classification problem with error-correcting output codes (ECOC) (Dietterich and Bakiri, 1995) is an approach that allows one to tackle multiclass classification problems using standard binary classification learning algorithms. The key idea is to represent each class with a multi-bit codeword. Then, the goal is to predict each bit for any given input by a binary classifier. If the combination of the outputs from the classifiers does not correspond to a legitimate codeword, the final output is chosen as the one that minimizes the Hamming distance—the number of bits needs to be changed. This procedure clearly takes the same direction as our inference with classifiers framework.

2.3 Inference Models

In this section, we introduce some frameworks that can be used to do inference. Due to tremendous varieties of inference models, we introduce here only those that will be used later in this dissertation.

2.3.1 Hidden Markov Model (HMM)

A hidden Markov model (HMM) is a probabilistic finite state automaton used to model the probabilistic generation of sequential processes. The model consists of a finite set $\mathcal{S}$ of states, a set $\mathcal{O}$ of observations, an initial state distribution $P_1(s)$, a state-transition distribution $P(s|s')$ for $s, s' \in \mathcal{S}$ and an observation distribution $P(o|s)$ for $o \in \mathcal{O}$ and $s \in \mathcal{S}$. A sequence of observations is generated as follows. An initial state is picked according to the initial state distribution $P_1(s)$. This state then produces an observation according to the observation distribution $P(o|s)$ and transits to a new state according to state-transition distribution $P(s|s')$. The new state subsequently produces the next observation, and this process goes on until it reaches a designated final state. This process

introduces the following two probabilistic assumptions for HMMs. First, given all the history, the current state depends only on the previous state:

$$P(s_t|s_{t-1}, s_{t-2}, \ldots, s_1, o_{t-1}, \ldots, o_1) = P(s_t|s_{t-1}). \tag{2.1}$$

Second, given all the history, the current observation depends only on the current state:

$$P(o_t|s_t, s_{t-1}, \ldots, s_1, o_{t-1}, o_{t-2}, \ldots, o_1) = P(o_t|s_t). \tag{2.2}$$

In order to use the model, there are three sets of parameters that need to be specified—the initial state probability $P_1(s)$, the transition probability $P(s|s')$, and the observation probability $P(o|s)$. Given that these parameters are unknown, they can be estimated in two scenarios. In the first scenario, only the observation sequences, $\mathbf{o} = \langle o_1, o_2, \ldots o_n \rangle$, can be observed. This can be regarded as *unsupervised* learning. The learner has to figure out how to reason about the unobserved state sequence. The most common way to deal with this is to find the parameters of the models that maximize the likelihood of observed sequences, $P(\mathbf{o})$. This can be done with Baum-Welch algorithm (Baum et al., 1970), which is an EM-type algorithm (Dempster, Laird, and Rubin, 1977) that gradually updates the parameters by iteratively using the current parameters to estimate the unobserved variables, and then maximizing the parameters based on the observed and estimated values. Since the algorithm improves the likelihood in each iteration, it guarantees to converge to a local minimum. On the other hand, in *supervised* learning, an observation sequence $\mathbf{o} = \langle o_1, o_2, \ldots o_n \rangle$ is supervised by a corresponding state sequence $\mathbf{s} = \langle s_1, s_2, \ldots s_n \rangle$. This allows one to estimate the most likely parameters, maximizing the likelihood, more directly, for example, by using the frequency count. As mentioned before, maximizing the likelihood in training lacks discriminative power. There exist also alternative approaches for training HMMs that attempt to overcome this issue (Kakade, Teh, and Roweis, 2002).

Given that the parameters are now specified, one can use HMMs to reason about the sequential

Viterbi Algorithm

Input: an observation sequence $\mathbf{o} = \langle o_1, o_2, \ldots, o_n \rangle$
Output: the most likely state sequence $\mathbf{s} = \langle s_1, s_2, \ldots, s_n \rangle$

1. Initialization:

   $\delta_1(s) \leftarrow P_1(s)P(o_1|s), \forall s \in \mathcal{S}$

2. Computation:

   For $t \leftarrow 2 \ldots n$
   $\delta_t(s) \leftarrow [\max_{s' \in \mathcal{S}} \delta_{t-1}(s')P(s|s')]\, P(o_t|s)$
   $\psi_t(s) \leftarrow \arg\max_{s' \in \mathcal{S}} \delta_{t-1}(s')P(s|s')$

3. Output:

   $s_n \leftarrow \arg\max_{s \in \mathcal{S}} \delta_n(s)$
   For $t \leftarrow n-1 \ldots 1$
   $s_t \leftarrow \arg\max_{s \in \mathcal{S}} \psi_{t+1}(s_t)$

Figure 2.1: Viterbi algorithm

data in various ways. In this dissertation, we focus only on one problem, i.e. finding the most likely state sequence given an observation sequence:

$$\mathbf{s} = \arg\max_{\mathbf{s} \in \mathcal{S}^n} P(\mathbf{s}|\mathbf{o}),$$

where n is the length of the sequence. This problem can be solved efficiently using a dynamic programming method which is called Viterbi algorithm (Viterbi, 1967). The algorithm is summarized in Figure 2.1.

The algorithm is built based on the independence assumption of HMMs which allows one to compute dynamically the best state sequences up to each time step. Specifically, given an observation $\mathbf{o} = \langle o_1, o_2, \ldots, o_n \rangle$, we define:

$$\delta_t(s) = \max_{s_1, s_2, \ldots, s_{t-1}} P(s_1, s_2, \ldots, s_{t-1}, s_t = s, o_1, o_2, \ldots, o_t).$$

This is the highest probability of any sequences generated by the model up to time t with the restriction that the state at time t takes the value s, and the observation sequence are specified as given. This value can be recursively computed by

$$\begin{aligned}
\delta_t(s) &= \max_{s_1,s_2,\ldots,s_{t-1}} P(s_1, s_2, \ldots, s_{t-1}, s_t = s, o_1, o_2, \ldots, o_t) \\
&= \max_{s_1,s_2,\ldots,s_{t-1}} P(s_t = s, o_t | s_1, s_2, \ldots, s_{t-1}, o_1, o_2, \ldots, o_{t-1}) \\
&\qquad P(s_1, s_2, \ldots, s_{t-1}, o_1, o_2, \ldots, o_{t-1}) \\
&= \max_{s_1,s_2,\ldots,s_{t-1}} P(o_t | s_t = s) P(s_t = s | s_{t-1}) P(s_1, s_2, \ldots, s_{t-1}, o_1, o_2, \ldots, o_{t-1}) \\
&= P(o_t | s_t = s) \max_{s' \in \mathcal{S}} P(s_t = s | s_{t-1} = s') \\
&\qquad \max_{s_1,s_2,\ldots,s_{t-2}} P(s_1, s_2, \ldots, s_{t-1} = s', o_1, o_2, \ldots, o_{t-1}) \\
&= P(o_t | s_t = s) \max_{s' \in \mathcal{S}} P(s_t = s | s_{t-1} = s') \delta_{t-1}(s') \\
&= \left[\max_{s' \in \mathcal{S}} \delta_{t-1}(s') P(s | s') \right] P(o_t | s).
\end{aligned}$$

Hence, the most likely state sequence is the one that induces the probability:

$$P^* = \max_{s \in \mathcal{S}} \delta_n(s).$$

For a comprehensive tutorial on HMMs, see the paper by Rabiner (1989). Different extensions to the standard HMMs have also been studied in order to fit better different real world applications, Bengio (1999) provides a good survey on these models.

2.3.2 Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) is defined as a triple $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$. $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is a finite set of *variables*. $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$ is the set of *domains* where $\mathcal{D}_i$ contains the possible values that can be assigned to variable v_i. $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ is a set of *constraints*. Each constraint $\mathcal{C}_i$ is a pair $\langle \mathcal{S}_i, \mathcal{R}_i \rangle$ where $\mathcal{S}_i = \{v_{i_1}, v_{i_2}, \ldots, v_{i_r}\} \subseteq \mathcal{V}$ defines the scope of $\mathcal{C}_i$

and $\mathcal{R}_i \subseteq \mathcal{D}_{i_1} \times \mathcal{D}_{i_2} \times \ldots \times \mathcal{D}_{i_r}$ restricts the values that can be assigned to the variables in $\mathcal{S}_i$ at the same time. A solution to a CSP is an assignment to the variables that does not violate any constraints in $\mathcal{C}$. A Boolean CSP is a CSP with its domains are restricted to only Boolean values.

In general, CSP is a very hard problem since it is a very generic problem-solving framework that generalizes many hard combinatorial problems such as graph coloring and satisfaction problem. In fact, Mackworth (1977) showed that deciding whether a CSP has a solution or not is an NP-complete problem. The restriction in the domains to Boolean values does not make the problem easier. What dictates the complexity of the problem is mainly the structure of the constraints which can be visualized by a *constraint graph* that represents each variable as a node with edges connecting any two nodes if their corresponding variables are together in the scope of a constraint. For example, if the constraint graph is a tree, the problem can be solved in linear time.

Current research in CSP focuses on two aspects. In the first aspect, heuristics are invented in order to find an algorithm that can solve general CSP quickly in real world problems. In the other aspect, effort has been made in order to find a restriction to the structure of the problem such that the problem can be solved more efficiently without sacrificing much of its expressiveness.

For a comprehensive coverage of constraint satisfaction problem and its related issues, see the book by Dechter (2003).

2.3.3 Integer Linear Programming

A *linear programming* (LP) problem is a problem of solving for a solution that optimizes (either minimizes or maximizes) a linear objective function subject to a set of linear constraints. Formally, given a vector of variables, $\mathbf{z} = \langle z_1, \ldots, z_d \rangle$, a cost vector $\mathbf{p} \in \mathbb{R}^d$, and cost matrices $\mathbf{C}_1 \in \mathbb{R}^{t_1} \times \mathbb{R}^d, \mathbf{C}_2 \in \mathbb{R}^{t_2} \times \mathbb{R}^d$, where t_1 and t_2 are the numbers of inequality and equality constraints

and d is the number of variables, a linear program is defined as:

$$\begin{aligned} &\text{minimize (or maximize)} && \mathbf{p} \cdot \mathbf{z} \\ &\text{subject to} \\ & && \mathbf{C}_1\mathbf{z} \geq \mathbf{b}_1 \\ & && \mathbf{C}_2\mathbf{z} = \mathbf{b}_2 \\ & && \mathbf{z} \in \mathbb{R}^d \end{aligned}$$

The solution to the linear program is an assignment to $\mathbf{z}$ that minimizes (or maximizes) the objective function. The problem is *infeasible* if there is no solution that can completely satisfy the constraints.

Note that any linear program can be reduced into an equivalent problem in a *standard form*. In this form, the objective is to maximize a linear function, there are only inequality constraints, and all variables are non-negative. In addition, any linear program can also be reduced into an equivalent problem in a *slack form*, which is similar to the standard form, but has only equality constraints, by introducing some slack variables. Although, expressing the problem in the slack form is less intuitive, it allows one to solve the problem by using the well-known *simplex* algorithm which can run very efficiently in practice.

An *integer linear program* (ILP) is a linear program with an additional constraint that the values of the output variables are only integers. It is important to note that in case of Boolean variables, which is a special case of ILP, any Boolean constraints can be equivalently expressed with a set of linear constraints which makes ILP generally very powerful. However, ILP is much harder than LP and is known to be an NP-hard problem. Nevertheless, with current technology, commercial packages such as Xpress-MP (Dash Optimization, 2004) can handle a large number of variables and solve the problem quickly in practice.

Further reading on LP and ILP can be found in the book by Cormen et al. (2001a) and Schrijver (1986).

Chapter 3

Inference with Classifiers

A large number of problems, especially in natural language processing, involve outputs with complex structure resulting in very large number of possible outcomes which make the direct application of a multiclass learning practically infeasible. A natural approach to these problems is to separate the task into two levels. In the first, several (learned) classifiers are evaluated and output classifications along with their confidence values (or conditional probabilities). The second is the inference level—a process which takes into account the outcomes of the different classifiers along with some domain specific constraints, and infers the final outcome for the global problem in a way that is coherent with the constraints.

In this chapter, we formalize the framework of *inference with classifiers*, and explain how the structured output problems fit into this framework. In addition we provide a survey of inference models, and the learning approaches that have been used in this framework in the literature.

3.1 Inference with Classifiers Framework

Traditional classification problem involves learning a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ mapping from an input $\mathbf{x} \in \mathcal{X}$ in some domain to a single label $y \in \mathcal{Y}$ where $\mathcal{Y}$ is assumed to be finite. Many real word applications, however, do not involve only a single prediction but multiple of them that need to be done in some coherent way. We conceptualize the problem in two levels: 1) the classification level—where several different classifiers are learned and evaluated to output their confidences over possible predictions—and 2) the inference level—where a final, global, decision is made, based on the outcomes of these classifiers with respect to some domain knowledge encoded in the form

of global constraints over the mutual outputs of the local classifiers.

Formally, the problem deals with multi-label outputs $\mathbf{y} \in \mathcal{C}(\mathcal{Y})$ where $\mathcal{Y} = \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_l$ defines the space of sequences of output labels and $\mathcal{C}(\mathcal{Y}) \subseteq \mathcal{Y}$ defines the space of only legitimate outputs that satisfy some constraints. These constraints are also where the structure of the outputs are encoded. Throughout the dissertation, we use $\mathbf{y}_{\setminus i}$ and $\mathcal{Y}_{\setminus i}$ such that $\mathbf{y}_{\setminus i} \in \mathcal{Y}_{\setminus i}$ to represent the output and its output space with y_i and $\mathcal{Y}_i$ being omitted.

At the classification level, there are several classifiers

$$f_{i \in \{1,2,\ldots,l\}} : \mathcal{X} \times \mathcal{Y}_{\setminus i} \times \mathcal{Y}_i \rightarrow \mathbb{R},$$

that output a level of confidence for each of their possible class labels in $\mathcal{Y}_i$ given an input $\mathbf{x} \in \mathcal{X}$ and possibly the labels of other classifiers $\mathbf{y}_{\setminus i} \in \mathcal{Y}_{\setminus i}$. Note that although the formalism does not make any restriction on the use of labels from other classifiers, in practice, classifiers may totally disregard this or uses such labels only partially in some local context, e.g. within a fixed-size window. Different uses of these labels in the classifiers may affect the efficiency of the evaluation of the classifiers and the inference process. A classifier that completely ignores the use of the labels from other classifiers is regarded as a *non-interactive* classifier.

At the inference level, the outputs of the classifiers are used to infer the final decisions that satisfy some constraints through the inference process which is defined as the following.

$$\mathit{Inference} \equiv \arg\max_{\mathbf{y} \in \mathcal{C}(\mathcal{Y})} \mathit{score}(\mathbf{x}, \mathbf{y} | f_1, f_2, \ldots, f_l),$$

where score is some global objective function over $\mathbf{x}$ and $\mathbf{y}$ given a collection of classifiers $f_1, f_2 \ldots, f_l$.

Our global task is a function

$$h : \mathcal{X} \rightarrow \mathcal{C}(\mathcal{Y}) \equiv \mathit{Inference}.$$

As an example, in predicting noun phrases in a given sentence of length n, the problem may consist of $2n$ classifiers that output the confidences over the predictions at different positions in the sentences. The predictions include whether or not each position is a beginning and an end of a noun phrase. Formally, $f_{i\in\{1,3,\ldots,2n-1\}}(\mathbf{x}, y_i)$ outputs the confidences over the predictions whether or not the word $(i+1)/2$ is a beginning of a noun phrase, and $f_{i\in\{2,4,\ldots,2n\}}(\mathbf{x}, y_i)$ output the confidences over the predictions whether or not the word $i/2$ is an end. The inference process then takes into account the outputs from the classifiers, and infer the final prediction that is consistent with the problem-specific constraints, i.e., phrases do not overlap, and maximizes some global objective scoring function $score$. For example, $score(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{2n} f(\mathbf{x}, y_i)$ is the linear summation of the confidences output by classifiers.

In fact, classification and inference as we defined do not differ functionally. Both, i.e. f, and $score$, involve mapping some inputs to some measures of confidences. The difference, however, lies within their concepts where the classifiers are obtained through learning, and inference are obtained based on the knowledge about the problem encoded by human with very minimal learning if any.

The focus of this dissertation is to the study this *Inference with Classifiers* approach. We show the practical legitimacy of the approach by applying it to two important fundamental problems in natural language processing, i.e. shallow parsing and semantic role labeling. Our study mostly focuses on the inference level with the aim to show the usefulness of incorporating constraints into the inference procedure.

The rest of this chapter will discuss how learning and inference can be done in this framework.

3.2 Inference

The inference procedures that can be used to combine the outputs of classifiers can be viewed in two categories. In the first category, some probabilistic assumptions are made about the outputs of classifiers, e.g. the confidence values are assumed to be some conditional probabilities. In this

case, the goal of the inference is to output the solution that maximizes the conditional probability given an input. The other approaches, however, do not make explicit probabilistic assumptions about the outputs of classifiers, but view them as some forms of costs. The goal of the inference is, therefore, to minimize the overall costs of the final solutions.

In this section we give an overview of different inference approaches that have been used to combine the outputs of classifiers.

3.2.1 Probabilistic Inference

Generally probabilistic inference is the problem of inferring the answer to some probabilistic queries given some underlying probabilistic model assumed for the task. Particularly for our problem, the query is to find the most likely output $\mathbf{y}$ given an input $\mathbf{x}$, that is:

$$\mathbf{y} = \arg\max_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y}|\mathbf{x}),$$

where the probabilistic model $P(\mathbf{y}|\mathbf{x})$ is defined over the output of the classifiers. Constraints are usually incorporated indirectly into the parameters of the model such that the probability of illegitimate outputs is always zero. The easiness of incorporating the constraints into the inference, therefore, highly depends on the properties of constraints and the underlying model.

Probabilistic inference can be categorized by the graphical models—either directed or undirected graphs—used to defined the dependency assumptions.

Probabilistic Inference with Directed Graph Representations

A directed (acyclic) graphical model such as Bayesian network (Pearl, 1988) factors the joint probability of the variables $y_1, y_2, \ldots, y_l$ by

$$P(y_1, y_2, \ldots, y_l) = \prod_{i=1}^{l} P(y_i|Parent(y_i)),$$

where $Parent(y_i)$ is the set of the variables the are the parents of y_i in the graph.

A well-known example of a specific directed graphical model is the hidden Markov model (HMM) in a sequential inference that assumes only the dependency of the classifiers over those that are adjacent. Richer models have also been tried (McCallum, Freitag, and Pereira, 2000) with different models producing different inference algorithms. In most cases, the model results in the local classifiers which output the confidence values that can be interpreted as some probabilities. Usually, the outputs of classifiers are viewed as the conditional probability:

$$f_i(\mathbf{x}, \mathbf{y}_{\setminus i}, y_i) = P(y_i|\mathbf{x}, \mathcal{N}(\mathbf{y}_{\setminus i})),$$

where $\mathcal{N}(\mathbf{y}_{\setminus i})$ is some context of $\mathbf{y}_{\setminus i}$, and the inference process combines these numbers in some way to infer the overall conditional probability $P(\mathbf{y}|\mathbf{x})$ based on the underlying probabilistic model.

For example, the underlying probabilistic model of the projection based Markov model (PMM) for sequential inference as described in Chapter 4 assumes a specific property that the output of the classifier at each position depends probabilistically on the given input and its previous output. Formally, the assumption can be written as:

$$P(y_i|y_{i-1}, y_{i-2}, \ldots, y_1, \mathbf{x}) = P(y_i|y_{i-1}, \mathbf{x}).$$

In this case, each classifier is assumed to output:

$$f_i(\mathbf{x}, \mathbf{y}_{\setminus i}, y_i) = P(y_i|y_{i-1}, \mathbf{x}),$$

and the goal of the inference is to find the most likely output:

$$\begin{aligned}\mathbf{y} &= \arg\max_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x}) \\ &= \arg\max_{\mathbf{y}\in\mathcal{Y}} P(y_i|\mathbf{y}_{i-1}\mathbf{x}) \\ &= \arg\max_{\mathbf{y}\in\mathcal{Y}} \prod_{i=1}^{l} f_i(\mathbf{x}, \mathbf{y}_{\setminus i}, y_i)\end{aligned}$$

Inference with probabilistic directed graphical models has been used extensively in combining the outcomes of classifiers, e.g. the work by Cohen et al. (1992), McCallum, Freitag, and Pereira (2000), and Morgan and Bourlard (1990), because of its strong foundation from probability theories, its simplicity, and the modularity of classifiers resulting from the probabilistic assumptions of the models.

Probabilistic Inference with Undirected Graph Representations

Probabilistic inference with undirected graph is also known as Markov random field (MRF) (Li, 2001). Markov random field represents the joint probability of variables $\mathbf{y} = \{y_1, \ldots, y_l\}$ in the form of

$$P(\mathbf{y}) = Z^{-1} \exp\left(\sum_{c\in\mathcal{C}} U_c(\mathbf{y})\right),$$

where $\mathcal{C}$ define the set of all cliques in the graph, U_c is a potential function defined over the variables in the clique c, and Z^{-1} is a normalizing factor so that the probability sums to 1.

MRF can be used to model the dependency among the input and output variables by modeling

$$P(\mathbf{x}, \mathbf{y}) = Z^{-1} \exp\left(\sum_{c\in\mathcal{C}} U_c(\mathbf{x}, \mathbf{y})\right),$$

for some $\mathcal{C}$. Given that, the most likely assignment to $\mathbf{y}$ given $\mathbf{x}$ can be computed by

$$\begin{aligned}
\mathbf{y} &= \arg\max_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x}) \\
&= \arg\max_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{x},\mathbf{y}) \\
&= \arg\max_{\mathbf{y}\in\mathcal{Y}} Z^{-1} \exp\left(\sum_{c\in\mathcal{C}} U_c(\mathbf{x},\mathbf{y})\right) \\
&= \arg\max_{\mathbf{y}\in\mathcal{Y}} \exp\left(\sum_{c\in\mathcal{C}} U_c(\mathbf{x},\mathbf{y})\right) \\
&= \arg\max_{\mathbf{y}\in\mathcal{Y}} \sum_{c\in\mathcal{C}} U_c(\mathbf{x},\mathbf{y})
\end{aligned}$$

Unlike the directed graph approach, using undirected graphs usually does not result in local classifiers that directly have a probabilistic interpretation. Instead, the local classifiers are those that result from different factorization of the summation of the potential functions $\sum_{c\in\mathcal{C}} U_c(\mathbf{x},\mathbf{y})$ such that

$$\sum_{i=1}^{l} f_i(\mathbf{x},\mathbf{y}_{\setminus i},y_i) = \sum_{c\in\mathcal{C}} U_c(\mathbf{x},\mathbf{y}).$$

Hence,

$$\mathbf{y} = \arg\max_{\mathbf{y}\in\mathcal{Y}} \sum_{i=1}^{l} f_i(\mathbf{x},\mathbf{y}_{\setminus i},y_i). \tag{3.1}$$

Similar to general probabilistic approaches, the encoding of global constraints are usually done through the parameters of the models such that the probability of illegitimate solutions is zero. This usually makes the types of the constraints very restricted to those that occur within a clique. However, due to the linearity of the objective function of the inference, Roth and Yih (2005) formulates this inference problem as an integer linear program allowing general global constraints to be easily incorporated in terms of linear (in)equalities.

MRF has not been used in combining classifiers, but some of the work using MRF can also be viewed at the conceptual level as inference with classifiers such as the work by (Kleinberg and

Tardos, 1999). Another related example is Conditional Random Field (CRF) (Lafferty, McCallum, and Pereira, 2001) that is an inference approach based on MRF theory that has been recently developed and is gaining popularity in machine learning community.

3.2.2 Non-probabilistic Inference

In non-probabilistic inference, the outputs of classifiers are usually referred as costs, and the goal is of the inference is to find the assignment to the output $\mathbf{y}$ that maximizes or minimizes the overall cost. In this case, the scoring function in the inference is defined as the summation of the outputs of the classifiers:

$$score(\mathbf{x}, \mathbf{y}|f_1, f_2, \ldots, f_l) = \sum_{i=1}^{l} f_i(\mathbf{x}, \mathbf{y}_{\setminus i}, y_i).$$

Although, the approach sounds ad-hoc because there is no underlying theoretical justification, the inference, indeed, is functionally equivalent to that in the probabilistic inference with undirected graph (Equation 3.1).

In addition, if each output of the classifiers is equivalent to the probability of the output label given the input:

$$f_i(\mathbf{x}, \mathbf{y}_{\setminus i}, y_i) = P(y_i|\mathbf{x}).$$

Computing the scoring function is consequently equivalent to computing the expected number of correct labels for any assignment of $\mathbf{y}$, given the input $\mathbf{x}$. Hence, the goal of the inference:

$$\begin{aligned} \mathbf{y} &= \arg\max_{\mathbf{y} \in \mathcal{C}(\mathcal{Y})} \sum_{i=1}^{l} f_i(\mathbf{x}, \mathbf{y}_{\setminus i}, y_i) \\ &= \arg\max_{\mathbf{y} \in \mathcal{C}(\mathcal{Y})} \sum_{i=1}^{l} P(y_i|\mathbf{x}) \end{aligned}$$

is to find, for each input $\mathbf{x}$, the output $\mathbf{y}$ that maximizes the expected number of correct labels among those that satisfy the constraints $\mathcal{C}(\mathcal{Y})$.

The actual inference algorithm may be designed per specific type of constraints for efficiency.

However, in general, the problem can be viewed as an integer linear program (ILP) allowing an ILP solver to solve the problem with general constraints that can be simply encoded through linear (in)equalities.

3.3 Learning

The framework presents two possible scenarios for learning. The first scenario is to incorporate inference into learning algorithms. Incorporating the inference into learning may complicate the learning algorithm, but can lead to the hypothesis that directly optimizes the true global objective function of the problem, and, therefore, may improve the overall performance of the system in a given task. In the other scenario, classifiers are decoupled from the inference at the learning stage. Each learner is trained independently as a regular classification task. Only later at the evaluation, the inference is used to combine the outputs of the classifiers. This offers the simplicity of the training the classifiers.

This section presents a survey of the learning algorithms for these two scenarios.

3.3.1 Coupling Approach

The coupling approach integrates the inference into learning algorithms. In this case, global inference and learning become tightly coupled—learning is done under the assumed constraints. The advantage of this approach is that the learners are aware of the global inference, and hence, can adapt itself to the actual global task. At the same time, this also presents a potential disadvantage as it is no longer possible to assume that one learns several classifiers independently of the inference problem at hand, and then combines them, perhaps in different ways, at decision time, taking into account domain and tasks specific constraints. In addition, integrating inference into learning requires the availability of the training examples of the global problem which can be very expensive to obtain. Moreover, it usually requires longer training time since there is a need to perform the (slow) inference procedure many times at this training stage unless the inference allows some

decomposition such that performing the whole inference is not necessary.

In this section, we give a survey of three techniques that has recently been developed in the context of the coupling approach.

Discriminative Training of HMMs

Recall from Section 2.3.1, a Hidden Markov Model (HMM) consists of a finite set $\mathcal{S}$ of states, a set $\mathcal{O}$ of observations, an initial state distribution $P_1(s)$, a state-transition distribution $P(s|s')$ for $s, s' \in \mathcal{S}$ and an observation distribution $P(o|s)$ for $o \in \mathcal{O}$ and $s \in \mathcal{S}$. The independence assumptions assumed in HMMs allows us to compute the most likely state sequence $\mathbf{s}$, give an observation sequence $\mathbf{o} = \langle o_1, o_2, \ldots, o_n \rangle$ by

$$\begin{aligned}
\mathbf{s} &= \arg\max_{\mathbf{s} \in \mathcal{S}^n} P(\mathbf{s}|\mathbf{o}) \\
&= \arg\max_{\mathbf{s} \in \mathcal{S}^n} P(\mathbf{o}|\mathbf{s}) P(\mathbf{s}) \\
&= \arg\max_{\mathbf{s} \in \mathcal{S}^n} \prod_{t=2}^{n} \left[P(o_t|s_t) P(s_t|s_{t-1})) \right] P(o_1|s_1) P_1(s_1) \\
&= \arg\max_{\mathbf{s} \in \mathcal{S}^n} \sum_{t=2}^{n} \left[\log P(o_t|s_t) + \log P(s_t|s_{t-1}) \right] + \log P(o_1|s_1) + \log P_1(s_1) \qquad (3.2)
\end{aligned}$$

This calculation can be done efficiently using Viterbi algorithm. HMM has been widely used for sequential labeling problems by predicting the most likely state sequence given the observation.

Traditional approach for training HMMs is to maximize the likelihood of the training examples. Building on the observation that state estimation in HMMs is a linear function of its local parameters (Roth, 1999) as shown in Equation 3.2, an algorithm that globally learns the HMMs parameters is developed (Collins, 2002). The technique is based on applying the (voted) Perceptron learning algorithm to learn the parameters in a discriminative way, providing an advantage over the standard maximum likelihood estimation.

Precisely, define $u(o, s) = \log P(o|s)$, $v(s, s') = \log P(s|s')$, and $v(s) = \log P_1(s)$ with two types of indicator variables $\varphi_t^{\mathbf{s}}(s)$ and $\psi_t^{\mathbf{o}}(o)$ indicating whether s and o appear at time t of $\mathbf{s}$ and

$\mathbf{o}$, respectively. One may rewrite Equation 3.2 as

$$\begin{aligned}
\mathbf{s} &= \arg\max_{\mathbf{s}\in\mathcal{S}^n} \sum_{t=2}^{n} \left[u(o_t, s_t)\psi_t^{\mathbf{o}}(o_t)\varphi_t^{\mathbf{s}}(s_t) + v(s_t, s_{t-1})\varphi_t^{S}(s_t)\varphi_{t-1}(s_{t-1})\right] \\
&\quad + u(o_1, s_1)\psi_t^{\mathbf{o}}(o_t)\varphi_t^{S}(s_t) + v(s)\varphi_1^{\mathbf{s}}(s_1) \\
&= \arg\max_{\mathbf{s}\in\mathcal{S}^n} \sum_{t=2}^{n} \left[\sum_{s,s'} u(o_t, s_t)\psi_t^{\mathbf{o}}(o_t)\varphi_t^{\mathbf{s}}(s_t) + \sum_{o,s} v(s_t, s_{t-1})\varphi_t^{\mathbf{s}}(s_t)\varphi_{t-1}(s_{t-1})\right] \\
&\quad + \sum_{o,s} u(o_1, s_1)\psi_t^{\mathbf{o}}(o_t)\varphi_t^{\mathbf{s}}(s_t) + \sum_{s} v(s)\varphi_1^{\mathbf{s}}(s_1) \\
&= \arg\max_{\mathbf{s}\in\mathcal{S}^n} \sum_{s,s'} u(o_t, s_t) \sum_{t=1}^{n} \psi_t^{\mathbf{o}}(o_t)\varphi_t^{\mathbf{s}}(s_t) + \sum_{o,s} v(s_t, s_{t-1}) \sum_{t=2}^{n} \varphi_t^{\mathbf{s}}(s_t)\varphi_{t-1}(s_{t-1}) \\
&\quad + \sum_{s} v(s)\varphi_1^{\mathbf{s}}(s_1). \qquad (3.3)
\end{aligned}$$

With an appropriate index set i such that each possible value of tuples (o, s), (s, s'), and (s) is mapped to a unique index, one can further rewrite Equation 3.3 to

$$\mathbf{s} = \arg\max_{\mathbf{s}\in\mathcal{S}} \sum_{i=1}^{N} w_i \phi_i(\mathbf{o}, \mathbf{s}),$$

where N is the size of the index set, w_i represent $u(o, s)$, $v(s, s')$, and $v(s)$ corresponding to what i maps to, and likewise, $\phi_i(\mathbf{o}, \mathbf{s})$ represent $\sum_{t=1}^{n} \psi_t^{\mathbf{o}}(s)\varphi_t^{S}(s)$, $\sum_{t=2}^{n} \varphi_t^{\mathbf{s}}(s)\varphi_{t-1}^{S}(s')$, and $\varphi_1^{\mathbf{s}}(s)$ depending on i. Hence, learning algorithm for linear classifiers can be applied.

The inference problem as shown in Equation 3.2 can also be viewed as the problem of inference with classifiers. The right hand side of Equation 3.2 can be factorized into local prediction problems for each s_t based on the local observation o_t and the previous state s_{t-1}. The local classifier can be viewed as:

$$f_t(\mathbf{o}, s', s) = \log P(o_t|s) + \log P(s|s').$$

Given the classifiers, the inference problem is viewed as using the following scoring function.

$$score(\mathbf{o}, \mathbf{s}) = \sum_{t=1}^{n} f_t(\mathbf{o}, s_{t-1}, s_t).$$

Hence, the discriminative training of HMMs can also be used to train such local classifiers together.

In fact, the formulation can be generalized to any problem that can be modeled by the linear representation in the form of

$$h(\mathbf{x}) \quad = \quad \underset{\mathbf{y} \in \mathbf{GEN}(\mathbf{x})}{\arg\max} \sum_i w_i \phi_i(\mathbf{x}, \mathbf{y}),$$

where $\mathbf{GEN}(\mathbf{x})$ defines the constrained output space of $\mathbf{y}$, and ϕ_i is any feature function defined over $\mathbf{x}$ and $\mathbf{y}$. The method of discriminative training of HMMs can also be extended to this more general problem.

This also suggests that given that the local classifiers are represented by linear functions, and the scoring function of the inference is also linear, one can also apply the same technique as suggested by Collins (2002) to train the classifiers in a coupling fashion.

Conditional Random Field

The conditional random field method (CRF) is a probabilistic approach to sequential processes based on Markov random field (MRF) theory (Lafferty, McCallum, and Pereira, 2001). CRF utilizes similar representation as MRF, however, to represent the conditional probability $P(\mathbf{y}|\mathbf{x})$, instead of the joint probability. Formally,

$$P(\mathbf{y}|\mathbf{x}) = Z^{-1}(\mathbf{x}) \exp\left(\sum_{c \in \mathcal{C}} U_c(\mathbf{x}, \mathbf{y}) \right).$$

Specifically, CRF defines the interaction between output variables in the sequence only on those that are adjacent. Hence, the conditional probability can be written in the form of

$$\begin{aligned} P(\mathbf{y}|\mathbf{x}) &= Z^{-1}(\mathbf{x}) \exp\left(\sum_{i=1}^{l} U_i(y_i, y_{i-1}, \mathbf{x}) + V_i(y_i, \mathbf{x}) \right) \\ &= Z^{-1}(\mathbf{x}) \exp\left(\sum_{i=1}^{l} \sum_{k=1}^{n} \lambda_k u_k(y_i, y_{i-1}, \mathbf{x}) + \mu_k v_k(y_i, \mathbf{x}) \right), \end{aligned}$$

where u_k and v_k are feature functions and λ_k and μ_k are weights for their corresponding features. This assumes that the potential functions are linear. A variety of training algorithms have been developed to approximate this probability, e.g. Sha and Pereira (2003).

In evaluation, the model predicts the output sequence that maximizes the conditional probability given the input:

$$\begin{aligned}\mathbf{y} &= \arg\max_{\mathbf{y}\in\mathcal{Y}} Z^{-1}(\mathbf{x}) \exp\left(\sum_{i=1}^{l}\sum_{k=1}^{n} \lambda_k u_k(y_i, y_{i-1}, \mathbf{x}) + \mu_k v_k(y_i, \mathbf{x})\right) \\ &= \arg\max_{\mathbf{y}\in\mathcal{Y}} \sum_{i=1}^{l}\sum_{k=1}^{n} \lambda_k u_k(y_i, y_{i-1}, \mathbf{x}) + \mu_k v_k(y_i, \mathbf{x}).\end{aligned}$$

This can be done by a Viterbi-like dynamic programming.

Similar to the discriminative training of HMMs, Equation 3.4 can be viewed as combining classifiers defined as

$$f_i(\mathbf{x}, y_{i-1}, y_i) = \sum_{k=1}^{n} \lambda_k u_k(y_i, y_{i-1}, \mathbf{x}) + \mu_k v_k(y_i, \mathbf{x}),$$

with the scoring function

$$score(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{l} f_i(\mathbf{x}, y_{i-1}, y_i).$$

This suggests that as long as classifiers are linear with the interaction between the output variables limited to those that are sequentially adjacent, and inference is restricted to linear summation, parameter estimation techniques for CRF can be used to train the classifiers together.

The formulation of CRF allows incorporating constraints over the adjacent output variables. More general constraints would require derivations of new training algorithms which are not easy. Due to its linearity, the discriminative training technique for HMMs as described in previous section may also be applied to CRF; although its efficiency can be a big problem.

By separating learning and inference, one can incorporate the general constraints to CRF only at the evaluation time. This has been studied by Roth and Yih (2005), and the contribution of

general constraints to CRF even only at the evaluation time has been observed.

Max-Margin Markov Network

Max-Margin Markov network (M^3) (Taskar, Guestrin, and Koller, 2004) can be viewed as a generalization of CRF in terms of its representation where the output variables are not limited to a sequence. Instead a more general graph is permitted to define interaction between any pairs of variables. The decision function of M^3 can be written in the form

$$\mathbf{y} = \arg\max_{\mathbf{y}\in\mathcal{Y}} \sum_{(i,j)\in\mathcal{E}} \sum_{k=1}^{n} \lambda_k u_k(y_i, y_j, \mathbf{x}),$$

where $\mathcal{E}$ is the set of all edges in the graph. Obviously, when the edges connecting the output variables are reduced to a path, M^3 is equivalent to CRF.

What makes M^3 very interesting is in the derivation of the training algorithm that aims to maximize the margins. This is the similar type of regularization that makes support vector machine (SVM) powerful.

Similarly to CRF, M^3 may be viewed as combining classifiers defined as

$$f_i(\mathbf{x}, \mathbf{y}_{\setminus i}, y_i) = \sum_{k=1}^{n} \sum_{j|(i,j)\in\mathcal{E}} \lambda_k u_k(y_i, y_j, \mathbf{x}),$$

with the scoring function

$$score(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{l} f_i(\mathbf{x}, y_{i-1}, y_i).$$

Hence, training algorithms for M^3 can be used to train classifiers together given that the inference and the classifiers are linear with the limitation to only pairwise interaction between output variables.

3.3.2 Decoupling Approach

When the classifiers are trained without the knowledge on the inference, standard machine learning technique for classification problems may be used to train each classifier in order to make the local predictions.

Work along this line may differ in the choice of learning algorithms, and the inference which is mostly designed for task specific. Due to its popularity and flexibility, this section provides only a short survey and perspective on work that is related to the technical approaches presented in our work.

Previous work on the decoupling approach has been done mostly in the context of sequential processes but all with the goal of improving performance on a given task, rather than from a more general perspective. Most of the previous work along this line can be categorized into two groups. First, learning is intentionally split from the inference by the designers. However, the goal is usually to introduce classifiers in order to improve existing inference problem, not to introduce the inference to improve the predictions of the classifiers. In the other group, the inference models mathematically allow the estimation of the parameters to be done locally; hence, learning, e.g., estimation of parameters in probabilistic models, can take place locally and independently.

In the first case, several attempts to combine classifiers, mostly artificial neural networks (ANN), into Hidden Markov Models (HMM) have been made in speech recognition works over the last decade (Cohen et al., 1992; Morgan and Bourlard, 1990). The main goal in this line of work was to improve existing HMMs when it was clear that simply computing maximum likelihood estimates for the observation probabilities has weak discriminating power. Instead, these estimates were replaced by the outputs of ANN that were shown, theoretically and empirically, to be a good approximation of Bayesian a posteriori probabilities (Bourlard and Morgan, 1989; Bourlard and Wellekens, 1989; Richard and Lippmann, 1991). The outputs of the ANN is typically used directly in the HMM as the observation probability, either without Bayesian inversion at all or with a weak form of Bayesian inversion that uses the probability of states counting over all the training data

regardless of their positions An excellent summary of this approach in speech recognition can be found in a paper by Morgan and Bourlard (1995).

In a very similar manner, Brants (2000) exploited a way to improve HMMs in the context of part-of-speech tagging. In contrast to the standard HMMs, his TnT system used linear interpolation of unigram, bigram, and trigram to estimate the second-order transition probabilities and, more interestingly, used a Bayesian inversion to compute the observation probability for words that have not been seen in the training corpus. This probability was computed based only on the suffixes of the words. Specifically, the conditional probability of the POS tag given the suffixes was statistically estimated and a standard Bayesian inversion is used to compute the observation probability. Since all suffixes could be assumed seen in the training examples, simple frequency counts can be used to estimate the probability. The Bayesian inversion is then computed using the probability of states counting over all the training data while ours is computed through the assumed HMM model.

In the second category, different probabilistic models have been developed in which a learning process was used as the parameter estimation procedure for the models. Ratnaparkhi (1996), in his work on POS tagging, modeled the conditional probability of the tag sequence given the sentence as the product of the probability of each tag given its context. The context included the words and tags within a certain window. The parameter of this term was learned with the Maximum Entropy technique. Then, the global solution was found with a beam search algorithm. This is similar to works such as those by Roth and Zelenko (1998) and Toutanova, Klein, and Manning (2003) where similar approaches were used for each POS tag (with different kinds of classifiers) and it was found that, in the context of POS tagging, global inference over the classifiers makes little or no difference.

McCallum, Freitag, and Pereira (2000) suggested a similar sequential labeling technique to that of Ratnaparkhi's in the context of information extraction. Their Maximum Entropy Markov Model (MEMM) used a maximum entropy framework to learn classifiers which considered the input document with the previous labels as the information source. This, therefore, could be considered

as an instantiation of Ratnaparkhi's technique where only one previous label is used as context. This somewhat more restricted model, however, made possible the use of dynamic programming to infer the global solution efficiently while Ratnaparkhi's technique needs beam search for its estimation.

Although, the decoupling approach may potentially lack the benefit the coupling approach have by being able to adapt the classifiers to the global constraints, this disadvantage is not actually observed in real world applications. On the other hand, because of its modularity, decoupling the training from the inference is conceptually simpler and more efficient. In addition, there is no need for the training examples for the global task; hence, it allows incorporating previously designed and learned classifiers into the systems. Due to this advantages, decoupling is by far the more popular method.

For the similar reason, the decoupling approach is the method that is used to train all systems presented in this dissertation for various applications.

Chapter 4

Application I: Shallow Parsing

Shallow parsing is the task to identify syntactic phrases, such as noun phrases, in natural langauge sentences. It is studied as an alternative to the more expensive full-sentence parsing. This shallow analysis has been found useful in many large-scale language processing applications including information extraction and text summarization (Grishman, 1995; Appelt et al., 1993).

In this chapter, we develop three *inference with classifiers*-based approaches to address this problem. The first two approaches are based on a probabilistic model within a Markovian framework. In this case, classifiers are functions of the observation sequence and their outcomes represent states. We study two Markovian models that are used as inference procedures. The two models differ in the types of classifiers and the details of the probabilistic modeling. The other approach extends a constraint satisfaction formalism to deal with variables that are associated with costs which can be used to model the problem of inference with classifiers. In this approach general constraints can be incorporated flexibly and algorithms can be developed that closely address the true global optimization criterion of interest. For all approaches we develop efficient inference algorithms that use general classifiers to yield the inference. All three approaches are evaluated experimentally on the tasks of identifying base noun phrases (NP) and subject-verb (SV) patterns.

Section 4.1 gives the definition of the task, specifically the definition of NPs and SVs. Section 4.2 presents how the problem can be modeled as an inference with classifier problem. Then, the three different inference approaches are introduced in Section 4.3, 4.4, and 4.5. Finally, experimental evaluations are presented in Section 4.6.

4.1 Shallow Parsing Task Definition

Shallow parsing tasks involve the identification of phrases or words that participate in a syntactic relationship. In this work we study the identification of two classes of phrases, base Noun Phrases (NP) and Subject Verb (SV) patterns. These two classes differ significantly in their statistical properties and thus studying both classes allows us to study the robustness of our methods to several assumptions.

A base NP is defined following the widely accepted definition presented by Ramshaw and Marcus (1995) as a non-recursive noun phrase that includes determiners but excludes post-modifying prepositional phrases or clauses. For example, two base NPs are marked by the pairs of brackets in the following part of a sentence:

```
...  presented [ last year ] in [ Illinois ] in front of ...
```

An SV pattern suggested by Argamon, Dagan, and Krymolowski (1999) is defined as a phrase starting with the subject of the sentence and ending with the first verb, excluding modal verbs[1]. Sample SV patterns are bracketed in the following:

```
[ The theory presented claims ] that [ the algorithm runs ] and
                        performs ...
```

As in previous work, the evaluation is concerned with identifying non-overlapping NPs and SVs.

4.2 System Modeling

Shallow parsing as discussed here can be viewed as identifying non-overlapping phrases in sentences. A straightforward approach is to train a classifier to predict for each possible phrase in the

[1] According to this definition the identified verb may not correspond to the subject, but this phrase still contains meaningful information; in any case, the learning method presented is independent of the specific definition used.

sentence whether or not it is a phrase of interest. For example, to identify base noun phrases, the classifier may predict:

`John sees a unicorn .`

$$\begin{array}{llll} \langle \texttt{John} \rangle & \mapsto \text{NP} & \langle \texttt{John sees} \rangle & \mapsto NULL \\ \langle \texttt{a unicorn} \rangle & \mapsto \text{NP} & \langle \texttt{sees a} \rangle & \mapsto NULL \\ & & \ldots & \mapsto NULL \end{array}$$

This, however, requires a quadratic number of classifiers' predictions for each sentence. Very common to the shallow parsing literature, an alternate approach is to define an intermediate representation that can equivalently represent the solutions of the problem and can be evaluated more efficiently.

An example is the BIO representation (Ramshaw and Marcus, 1995) where each input symbol is labeled by **I**, **O**, or **B** to represent **I**nside, **O**utside, and **B**eginning of a phrase. **B** is necessary to distinguish when two consecutive phrases occurs. As in the previous example of `John sees a unicorn .`, the solution is

`John`	`sees`	`a`	`unicorn`	`.`
B	O	B	I	O

A variation such as IOE representation where one may predict, instead of **B**, **E**nd of a phrase is also possible. Following this, for the previous example, the classifier should output.

`John`	`sees`	`a`	`unicorn`	`.`
E	O	I	E	O

Other variations also exist; however, through out this work we assume only these two types of classifiers, i.e. BIO and IOE. We would like to note here that the choice of representations does not generally dictate any significant difference in the performance at least in the context of shallow parsing used in our experiment (Tjong Kim Sang and Veenstra, 1999; Tjong Kim Sang, 2000). The choices we make here is due to the possibility to use the same representations across different inference models providing a fair comparison among them.

By combining BIO and IOE representations into a single sequence, the task now can be modeled as making alternative predictions between BIO and IOE along the words in the sentence as shown in the following example.

John	sees	a	unicorn	.
B E	O O	B I	I E	O O

The goal of the inference is to ensure that the final predictions are legitimate sequences which are those that follow the state diagram shown in Figure 4.1

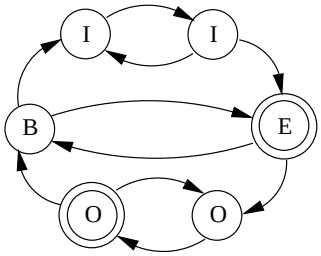

Figure 4.1: State-transition diagram constraining the outputs of shallow parsing system

4.3 Hidden Markov Model Inference

One may use HMMs to address the sequential labeling problem directly. However, as we will show in the experiments, this does not result in good performance. The main problem is due to the weakness in the direct approximation of observation probability. Instead, we suggest here to incorporate HMMs as an inference procedure and use it to combine the outcomes of more expressive classifiers.

In this section we introduce our first probabilistic model based inference. Specifically, the probabilistic model we use is Markovian. The confidence outputs of the classifiers will be viewed as a probability distributions over states in the Markov model, given an observation. The constraints in this case are encoded in the form of state-transition probabilities and the goal of the inference in this approach is simply to output the most likely state sequence for the outputs of classifiers,

respecting the constraints. These constraints can be incorporated into the HMM by constraining the state transition probability distribution $P(s|s')$. For example, set $P(s|s') = 0$ for all s, s' such that the transition from s' to s is not allowed. This encoding may be done explicitly or it might be recovered implicitly from training data if the data does not contain any illegal sequences. When completely learned, the model then outputs a phrase structure represented by the most likely state sequence.

In order to use an HMM as a inference procedure over the outputs of classifiers, we assume that the confidence supplied by the classifier is the probability of a state given the observation $P_t(s|o_t)$ where t is the time step or the order of the symbol in the sequence. This information can be used in the HMM framework by applying the Bayes rule to compute

$$P_t(o_t|s) = \frac{P_t(s|o_t)P_t(o_t)}{P_t(s)} \tag{4.1}$$

where $P_t(o)$ and $P_t(s)$ are the probabilities of observing o and being at s at time t, respectively. That is, instead of estimating the observation probability $P(o|s)$ directly from training data, we compute it from the classifiers' output. Although, doing so may empirically result in $P_t(o_t|s)$ that is not a legitimate probability distribution (summation is not equal to 1), it is important to note here that our goal is to provide a sound mechanism to combine outcomes of classifiers, not directly to compute the probability.

Notice also that normally in HMMs, the probability distributions are assumed to be stationary; that is, the observation probability $P(o|s)$ does not depend on t. However, we do not require this assumption here. The only stationary assumption we assume is for the state transition probability $P(s|s')$. We will add a subscript t to distributions that do not require this assumption.

$P_t(s)$, the probability of being at state s at time t can be calculated through the summation of

the probabilities of all possible sequences having s at time t. That is

$$\begin{aligned} P_t(s) &= \sum_{\ldots,s_{t+1},s_{t-1},\ldots,s_1} P(\ldots,s_{t+1},s,s_{t-1},\ldots,s_1) \\ &= \sum_{\ldots,s_{t+1},s_{t-1},\ldots,s_1} \ldots P(s_{t+1}|s)P(s|s_{t-1})P(s_2|s_1)P_1(s_1) \qquad (4.2) \\ &= \sum_{\ldots,s_{t+1}} \ldots P(s_{t+1}|s) \sum_{s_{t-1},\ldots,s_1} P(s|s_{t-1}) \ldots P(s_2|s_1)P_1(s_1) \qquad (4.3) \\ &= \sum_{s_{t-1},\ldots,s_1} P(s|s_{t-1}) \ldots P(s_2|s_1)P_1(s_1) \qquad (4.4) \end{aligned}$$

(4.2) comes directly from Markov assumption. In (4.3), the product of the terms in the first summation is basically the probability of being in a particular state sequence after time t given that s is the state at time t. Hence, the summation of this product over all possible assignments to the states after time t is 1 which brings us to (4.4). Given this one can, therefore, recursively compute $P_t(s)$ by

$$\begin{aligned} P_t(s) &= \sum_{s_{t-1}} P(s|s_{t-1}) \sum_{s_{t-2},\ldots,s_1} P(s_{t-1}|s_{t-2}) \ldots P(s_2|s_1)P_1(s_1) \\ &= \sum_{s_{t-1}} P(s|s_{t-1})P_{t-1}(s_{t-1}) \end{aligned}$$

where $P_1(s)$ and $P(s|s_{t-1})$ are the two required distributions for the HMM. In order to compute $P_t(o|s)$ in (4.1) we still need $P_t(o)$ which is harder to approximate. Fortunately, for each t we can treat it as a constant η_t because our goal is only to find the most likely sequence of states for given observations which are fixed for all compared sequences. Note that while $P_t(o)$ is hard to compute (and not needed), a clean probabilistic interpretation of this method relies on the classifier $P_t(s|o_t)$ producing outputs such that there exist values $P_t(o)$ so that $\sum_{o_t} P_t(s|o_t)P_t(o_t) = P_t(s)$.

With this scheme, we can still combine the classifiers' predictions by finding the most likely sequence for a given observation using standard dynamic programming (Viterbi). To do so, we incorporate the classifiers' outcomes in the algorithm's recursive step by computing $P_t(o_t|s)$ as

discussed above, and we choose the most likely state at time t to be:

$$\begin{aligned}
\delta_t(s) &= \max_{s' \in \mathcal{S}} \delta_{t-1}(s') P(s|s') P_t(o_t|s) \\
&= \max_{s' \in \mathcal{S}} \delta_{t-1}(s') P(s|s') \frac{P_t(s|o_t) P_t(o_t)}{P_t(s)} \\
&= \max_{s' \in \mathcal{S}} \delta_{t-1}(s') P(s|s') \frac{P_t(s|o_t) \eta_t}{P_t(s)}.
\end{aligned}$$

This is derived from the basic HMM independence assumptions as shown in (2.1) and (2.2). However, the above formulation, which uses the classifier outputs $P_t(s|o_t)$, opens the door for extending the notion of an observation. Indeed, in our experiment we estimate instead $P_t(s|\tilde{o}_t)$ where $\tilde{o}_t$ may contain more than only a single observation, e.g. the observations within a certain window around t or even the whole observation sequence. This significantly improves the phrase identification performance.

To use the HMM inference, we require, first, to have classifiers that output for each input symbol the probability of each possible state for that position. In practice, a classifier that does not specifically output this probability but some other confidence measure may still be used by normalizing the confidences of all states for each position to sum up to one. Then, the HMM inference process outputs the most likely state sequence based on the probabilities of each state given an observation supplied by the classifiers. When the state-transition probability is derived or learned properly, the imposed constraints over the order of states is implicitly encoded, and hence, the final output is guaranteed to satisfy the constraints. Note that although, in the experiment, the classifiers for the HMM approach are incorporated as described above, our preliminary experiment did not reveal any significant need of the denominator term in (4.1).

4.4 Projection based Markov Model Inference

In this section, we introduce another probabilistic inference that can be viewed as an extension of the previous HMM model. In HMMs, observations are allowed to depend only on the current state

and long-term dependencies among the observations are not modeled. Equivalently speaking, from the constraints point of view, the constraint structure is restricted to having a stationary probability distribution of a state given the previous one. Our second inference attempts to relax this by moving from a generative model to a *conditional* model, in which states are modeled directly, and the distribution of a state is allowed to depend both on the observation and the previous state. Formally, unlike the standard HMM, we make the following independence assumption:

$$P(s_t|s_{t-1}, s_{t-2}, \ldots, s_1, o_t, o_{t-1}, \ldots, o_1) = P(s_t|s_{t-1}, o_t).$$

Thus, given an observation sequence $\mathbf{o} = \langle o_1, o_2, \ldots, o_n \rangle$, we can find the most likely state sequence $\mathbf{s}$ by maximizing

$$\begin{aligned} P(\mathbf{s}|\mathbf{o}) &= \prod_{t=2}^{n} [P(s_t|s_{t-1}, s_{t-2}, \ldots, s_1, o_t, o_{t-1}, \ldots, o_1)] P_1(s_1|o_1) \\ &= \prod_{t=2}^{n} [P(s_t|s_{t-1}, o_t)] P_1(s_1|o_1). \end{aligned}$$

Hence, this model generalizes the standard HMM by combining the state-transition probability and the observation probability into one function. The most likely state sequence can still be recovered using the dynamic programming (Viterbi) algorithm if we modify the recursive step as the following:

$$\delta_t(s) = \max_{s' \in \mathcal{S}} \delta_{t-1}(s') P(s|s', o_t).$$

In this model, the classifiers' decisions are incorporated in the terms $P(s|s', o)$ and $P_1(s|o)$. Therefore, the classifiers have to take into account the current input symbol and, in addition, the previous state. The hope is that these new classifiers have better performance than the classifiers used in the HMM inference because they are given more information—the previous state—thus leading to improvement of the whole system.

We call this method the Projection based Markov model (PMM) inference since in learning these classifiers we follow the projection approach (Valiant, 1998) and separate $P(s|s', o)$ to many functions $P_{s'}(s|o)$ according to the previous states s'. Intuitively, this should give better classifiers since each of them will learn a simpler concept, in a smaller instance space restricted by the previous state. Hence, as many as $|\mathcal{S}|$ classifiers, projected on the previous states, are separately trained.

As before, the question of what constitutes an observation is an issue. However, unlike in HMMs, where considering an *observation window* in the observation sequence that is wider than the observation at time t, is a violation of the model, in PMMs it is not. The dependency assumption on the observation can always be modified to capture a wider window. In fact, the dependency assumption can be relaxed so that a state depends on the previous state and any part of the observation sequence:

$$P(s_t|s_{t-1}, s_{t-2}, \ldots, s_1, \mathbf{o}) \quad = \quad P(s_t|s_{t-1}, \tilde{o}_t).$$

This will result in a slight generalization of the model:

$$P(\mathbf{s}|\mathbf{o}) \quad = \quad \prod_{t=2}^{n} [P(s_t|s_{t-1}, \tilde{o}_t)] P_1(s_1|\tilde{o}_1),$$

and, consequently,

$$\delta_t(s) \quad = \quad \max_{s' \in \mathcal{S}} \delta_{t-1}(s') P(s|s', \tilde{o}_t).$$

Indeed, in our experiment, we show the contribution of estimating $P_{s'}(s|o)$ using a wider window in the observation sequence.

The requirements for using the PMM inference are similar to those for the HMM inference. First, for each input symbol and each possible previous state, the classifiers need to output the probability of the current state. Again, in practice, a classifier that outputs another confidence

measure may be used via the normalization of the confidences. Then, the PMM inference can be applied on top of these classifiers to output the most likely state sequence. With the properly derived or learned state-transition probability, the output will be guaranteed to satisfy the imposed constraints.

4.5 Constraint Satisfaction Based Inference

This section describes a different model developed to infer the phrase structure of the input sentence given local classifiers. The approach is based on an extension of the Boolean constraint satisfaction formalism (see Section 2.3.2) to handle variables that are outcomes of classifiers. The section begins by introducing the extension of Boolean constraint satisfaction problem (CSP) to handle probabilistic variables. Then, we show how this can be used to develop an inference procedure.

4.5.1 An Extension to Boolean Constraint Satisfaction Problem

A Boolean constraint satisfaction problem is defined over a set of variables, a set of values for each variable, and a set of constraints. Specifically, it consists of a set of n variables $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, each ranges over values in $\mathcal{D}_i = \{0, 1\}$. A constraint is a relation over a subset of variables, defining a set of allowed *global* assignments to the corresponding variables. A solution to a CSP is an assignment that satisfies all constraints. In fact, the goal of a CSP might be to find just one of the solutions, all solutions, or only the one that optimizes some objective function (the optimization problem), or to decide if there exists a solution at all.

For shallow parsing, the goal is to output only the optimal solution, requiring that we define an appropriate objective function for this problem. In order to do so, we extend the CSP formalism to deal with variables that are associated with costs); the sought after solution will be the one that minimizes the overall cost function. Specifically, given a CSP containing a set $\mathcal{V}$ of n variables, we additionally associate with each variable a cost function $c_i : \mathcal{D}_i \rightarrow \mathbb{R}$. The problem is to find

the solution that satisfies the constraints and, in addition, minimizes the overall cost

$$c \;=\; \sum_{i=1}^{n} c_i(v_i).$$

Our problem has only Boolean variables. Hence, the constraints can be put together as a Boolean formula in a conjunctive normal form (CNF). In the next section, we will show how the shallow parsing problem can be cast into this CSP with Boolean variables. With an appropriate modeling, the CSP looks very intuitive, and can be solved efficiently.

4.5.2 Constraint Satisfaction with Classifiers (CSCL)

As before, relying on a concrete representation of phrases imposes a set of constraints over the representation. This constraints can be formulated as a CSP in two ways. The first is the direct transformation from the state-diagram, that associates a variable with each possible prediction at each symbol location. This formulation gives a linear (in the input sentence) number of Boolean variables and a fairly straightforward way to encode the constraints. For example, if the variable representing the **B**eginning at location k is on, then at least one of the **E**nd variables at locations $> k$ must be on. It is also clear how to derive a CNF formula conjuncting all the constraints. The natural cost function is the probability supplied by the classifiers. However, this approach has two problems: 1) the constrains are not binary, yielding a k-CNF formula f with $k \geq 3$, for which the decision problem is computationally hard (Cook, 1971), and 2) the optimal solution does not optimize the global criterion—the confidence in the phrase structure.

Instead, we use this general scheme to encode phrases, rather then classifier outcomes, as variables. This yields a quadratic number of variables, but the constraints become binary constraints, encoding the restriction that phrases do not overlap. A satisfying assignment for the resulting 2-CNF formula can therefore be computed in polynomial time.

Formally, the problem is defined as follows. Given an input $\mathbf{o} = \langle o_1, o_2, \ldots, o_l \rangle$, the CSP consists of a set of variables $\mathcal{V} = \{\, v_{i,j} \,|\, 1 \leqslant i \leqslant j \leqslant l \,\}$. Each variable $v_{i,j}$ corresponds to a

potential phrase starting from the position i to the position j in the input. In addition, a cost $c_{i,j}$ is associated with each variable $v_{i,j}$. The constraints can be written as a Boolean formula. For instance, in the problem of identifying non-overlapping phrases, the constraints are such that for any two phrases that overlap, at most one of them may be included in the solution. This can be summarized in the following CNF formula:

$$f = \bigwedge_{\pi^{a,b} \text{ overlaps } \pi^{c,d}} (\neg v_{a,b} \vee \neg v_{c,d}),$$

where $\pi^{i,j}$ represents the phrase from i to j.

In general, the corresponding optimization problem is still NP-hard. In the restricted case that the cost for each variable is within $[0, 1]$, there is an efficient algorithm that is guaranteed to find a satisfying assignment with cost that is at most twice the optimal (Gusfield and Pitt, 1992). However, for the specific case of our non-overlapping phrase identification, the problem can be solved efficiently, and the optimal solution can be computed in time that is linear in the number of possible phrases (quadratic in the length of the input).

We present the solution by constructing a directed acyclic graph (DAG) that correspond to the problem and showing that the solution to the optimization problem corresponds to a shortest path in it. The graph is constructed on the observation symbols, with legitimate phrases (the variables of the CSP) as its edges and their costs as the edges' weights. The corresponding graph is constructed with the procedure shown in Figure 4.2.

The construction of this DAG takes quadratic time and corresponds to constructing the 2-CNF formula above. It is easy to see that each path in this graph corresponds to a legitimate set of phrases (with no overlap), and consequently, that the shortest path in this graph corresponds to the solution with minimal cost. The time complexity of this algorithm is linear in the number of phrases, precisely $\Theta(\mathcal{V}' + \mathcal{E}')$, by using the DAG-Shortest-Paths algorithm (Cormen et al., 2001b).

The main problem in this approach is how to determine the cost c as a function of the confidence given by the classifiers. There can be many possible choices of cost functions based on the

Graph Construction

Given the list of all possible phrases $\mathcal{V}$, a weighted graph, $G = (\mathcal{V}', \mathcal{E}')$, is created such that every path from *source* to *sink* in the graph corresponds to a possible solution.
The vertex set $\mathcal{V}'$ consists of:

- B, the set of all positions that are the beginning of a phrase
- E, the set of all positions that are the end of a phrase
- $source$, a beginning vertex
- $sink$, an ending vertex

The edge set $\mathcal{E}'$ consists of:

- $(source, b)$, for all $b \in B$.
- $(e, sink)$, for all $e \in E$.
- (b, e), for all phrases beginning at b and ending at e.
- (b, e), for all $e \in E$ and $b \in B$ such that e comes before b in the original sentence.

The weights are set by the cost function used.

Figure 4.2: Graph construction for reducing the non-overlapping phrase identification problem to a DAG shortest path problem

interpretation of the problem. Our experiments revealed, though, that the algorithm is robust to modification in the cost function. Next we present a cost function which has a nice interpretation, and was also shown to produce good experimental results.

Cost Function

An appropriate cost function should reflect the global goal of the phrase identification problem. For this problem, the performance is measured by recall and precision. Recall is the percentage of phrases that are correctly identified. In order to increase recall, we have to increase the number of correctly predicted phrases. Precision is the percentage of identified phrases that are indeed correct phrases. In order to increase precision, we have to decrease the number of phrases predicted incorrectly. Therefore, a good cost function should reflect these two aspects—maximizing the number

of correct phrases and minimizing the number of wrong phrases. The cost function introduced in this section captures the first part—that of maximizing the number of correct phrases. The missing second part can be overcome by a technique which we discuss later.

Let $x_{i,j}$ be a binary indicator variable which is equal to 1 if the phrase $\pi^{i,j}$ is a correct phrase, and 0 otherwise. Given a set of phrases $\widehat{\mathcal{V}}$, the number of correct phrases in this set is

$$\text{number of correct phrases} \quad = \quad \sum_{v_{i,j} \in \widehat{\mathcal{V}}} x_{i,j}.$$

Let the probability that the phrase $\pi^{i,j}$ is a correct phrase be $p_{i,j}$. Then the expected number of correct phrases in the given set of phrases is given by

$$\mathbf{E}[\,\text{number of correct phrases} \,|\, \widehat{\mathcal{V}}\,] = \sum_{v_{i,j} \in \widehat{\mathcal{V}}} p_{i,j}. \tag{4.5}$$

Therefore, we can maximize the expected number of correct phrases by finding the set of phrases that maximizes the terms in Equation 4.5. Equivalently, we can do that by finding:

$$\mathcal{V}^* \quad = \quad \arg\min_{\widehat{\mathcal{V}} \subseteq \mathcal{V}} \sum_{v_{i,j} \in \widehat{\mathcal{V}}} -p_{i,j}.$$

Thus a desired cost function for our shortest path problem is to assign each phrase with a cost that is negative of the probability of the phrase being correct:

$$c_{i,j}(v_{i,j}) \quad = \quad \begin{cases} -p_{i,j} & \text{if } v_{i,j} = 1 \\ 0 & \text{otherwise} \end{cases}.$$

Computing this probability for a phrase can be done based on assumptions made on the phrase structure. For example, the simplest case may be to assume independence among symbols in a phrase. In the case of shallow parsing task which we use for the experimental evaluation, it is reasonable to further assume that the important parts of a phrase are only its beginning and its end.

Based on this assumption, and given that we use the BIO and IOE representations, the probability of a phrase being correct may be defined as

$$p_{i,j} = P_i^B(\mathbf{B})P_j^E(\mathbf{E})$$

where $P_i^B(\mathbf{B})$ is the probability that the first symbol o_i in the phrase is actually the beginning of a phrase and $P_j^E(\mathbf{E})$ is the probability that the last symbol o_j in the phrase is actually the end of a phrase. These two probabilities are supplied by the classifiers.

Maximizing only the number of correct phrases causes the system to output a set of phrases that covers all symbols without caring that some of the phrases are more likely to be incorrect. Therefore, minimizing the number of wrong phrases is important. A simple way to do so is to set a threshold to filter out phrases with low probability. A proper setting of the threshold will result in a good tradeoff between the two goals of our optimization. In our experiments, instead of using one threshold to filter out phrases, we uses two thresholds, both tuned on a development set; one is used to filter out the low probability **B**s and the other to filter out the low probability **E**s. This can also reduce significantly the number of candidates in consideration, thus affecting the running time of the graph algorithm described before.

4.6 Experiment

In this section, we first describe our experimental methodology, then we explain the features used by the classifiers and, finally, we present our experimental results.

4.6.1 Methodology

We conducted experiments mainly to compare different inference models, namely the HMM, PMM and CSCL inference[2]. We experimented with both NPs and SVs and show the results for

[2]The software package of CSCL inference is available at http://l2r.cs.uiuc.edu/~cogcomp. A demonstration of this approach in the task of full chunking (Li and Roth, 2001) can also be found there.

two different feature sets for the classifiers—part-of-speech (POS) information only and POS with additional lexical information (words). The results of interest were the recall and precision as well as F_β, defined by

$$F_\beta = \frac{(\beta^2 + 1) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

(here $\beta = 1$). *Recall* is the percentage of correct phrases that are identified, and *precision* is the percentage of identified phrases that are indeed correct.

In addition to the regularized winnow update rule within SNoW, we also experimented with other classifiers. One was naive Bayes (NB, implemented within the SNoW architecture too) that, under certain assumptions, outputs conditional probabilities, which was a property we looked for. The other classifier was the *simple* maximum likelihood probability estimation—simply the normalization of frequency count. This *simple* classifier is used to provide a baseline to other classifiers, and was not expected to perform well.

The data sets used were the standard data sets for this problem (Argamon, Dagan, and Krymolowski, 1999; Muñoz et al., 1999; Ramshaw and Marcus, 1995; Tjong Kim Sang and Veenstra, 1999) taken from the Wall Street Journal corpus in the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993). For NPs, the training and test corpus was prepared from Sections 15 to 18 and Section 20, respectively, and tagged by Brill's POS tagger (Brill, 1995). The SV corpus was prepared from Sections 1 to 9 for training and Section 0 for testing, and their POS tags were taken as they were in the Treebank. The sizes of the training and test data are summarized in Table 4.1 and Table 4.2.

Data	Sentences	Words	NP Patterns
Training	8936	211727	54758
Test	2012	47377	12335

Table 4.1: Sizes of the NP training and test data sets

For the CSCL inference, there is a need to determine a set of thresholds to filter out phrases with low probability, in order to resolve the problem of incorporating too many phrases as described in

Data	Sentences	Words	SV Patterns
Training	16397	394854	25024
Test	1921	46451	3044

Table 4.2: Sizes of the SV training and test data sets

Section 4.5.2. This set of thresholds is obtained by first training the system only on 90% of the training data and then searching for the best set of thresholds by evaluating on the remaining 10% of the training set.

4.6.2 Features

The features used in our system are relational features over the sentence and the POS tag information, which can be defined by three parameters, $w-$, $w+$, and k. Specifically, features are POS tag conjunctions, word conjunctions, or mixed conjunctions. Mixed conjunctions contain the features that have both POS-tags and words mixed together. All conjunctions of size up to k and within a window that includes the $w-$ words before and $w+$ after the designated word are generated.

An example is shown in Figure 4.3 where $(w-, w+, k) = (3, 3, 3)$ for POS tags, $(2, 2, 2)$ for words, and $(2, 2, 2)$ for mixed features. In this example the word "how" is the designated word with POS tag "WRB". "()" marks the position of the current word (tag) if it is not part of the feature, and "(how)" or "(WRB)" mark the position of the current word (tag) if it is part of the current feature. The distance of a conjunction from the current word (tag) can be induced by the placement of the special character "_" in the feature.

We experimented with a variety of parameter sets, but reported here only the best set. For this, all classifiers used features with $(w-, w+, k)$ equal to $(3, 3, 3)$ for POS tags, $(2, 2, 2)$ for words, and $(2, 2, 2)$ for mixed features. Again, this set of parameters were obtained experimentally by randomly dividing training data into 90% training set and 10% testing and tuning.

					⇓			
This	is	an	example	of	how	to	generate	features	.
DT	VBZ	DT	NN	IN	WRB	TO	VB	NNS	.

Conj. Size	1	2	3
POS-Tag Features	DT _ _ ()	DT NN _ ()	DT NN IN ()
$(w-, w+, k)$	NN _ ()	NN IN ()	NN IN (WRB)
$(3, 3, 3)$	IN ()	IN (WRB)	IN (WRB) TO
	(WRB)	(WRB) TO	(WRB) TO VB
	() TO	() TO VB	() TO VB NNS
	() _ VB	() _ VB NNS	
	() _ _ NNS		
Word Features	example _ ()	example of ()	
$(w-, w+, k)$	of ()	of (how)	
$(2, 2, 2)$	(how)	(how) to	
	() to	() to generate	
	() _ generate		
Mixed Features		NN of ()	
$(w-, w+, k)$		example IN ()	
$(2, 2, 2)$		IN (how)	
		of (WRB)	
		(WRB) to	
		(how) TO	
		() TO generate	
		() to VB	

Figure 4.3: An example of feature extraction. () represents the position of the target word.

4.6.3 Results

For each model we study three different types of classifiers. The *simple* classifier corresponds to the standard HMM in which $P(o|s)$ is estimated directly from the data, and the features are generated only from the target input symbol. Notice that when the observations are in terms of lexical items, these estimates are too sparse to be of any significance, so we leave these entries empty. The NB and SNoW classifiers use the same feature set as described before.

Table 4.3 and Table 4.4 summarize the results of the experiments on the NP and SV patterns respectively. We compare our results with those by Ramshaw and Marcus (1995) (RM95) who originated the NP dataset, Argamon, Dagan, and Krymolowski (1999) (ADK99) who originated the SV dataset, and the state-of-the-art results on NP and SV by Collins (2002) (Collins02) and

Method		POS tags only			POS tags+words		
Model	Classifier	Recall	Prec.	$F_{\beta=1}$	Recall	Prec.	$F_{\beta=1}$
HMM	SNoW	91.49	91.53	91.51	93.85	93.46	93.65
	NB	90.86	89.87	90.36	93.28	92.08	92.67
	Simple	89.08	86.62	87.83	—	—	—
PMM	SNoW	92.04	91.77	91.90	94.28	94.02	94.15
	NB	90.72	89.75	90.23	92.34	92.22	92.78
	Simple	59.72	63.27	61.44	—	—	—
CSCL	SNoW	90.99	91.89	91.44	94.12	93.45	93.78
	NB	88.97	90.34	89.65	91.28	92.88	92.08
	Simple	63.47	47.63	54.42	—	—	—
RM95		90.7	90.5	90.6	92.3	91.8	92.0
ADK99		91.6	91.6	91.6	—	—	—
MPRZ99		90.9	90.3	90.6	93.1	92.4	92.8
Collins02		—	—	—	—	—	93.63

Table 4.3: Results of different methods on NP recognition

Method		POS tags only			POS tags+words		
Model	Classifier	Recall	Prec.	$F_{\beta=1}$	Recall	Prec.	$F_{\beta=1}$
HMM	SNoW	81.47	71.93	76.40	86.43	75.69	80.71
	NB	83.48	69.64	75.93	87.39	69.93	77.69
	Simple	75.36	56.91	64.85	—	—	—
PMM	SNoW	83.34	88.86	86.01	87.32	94.22	90.64
	NB	79.70	75.48	77.53	85.74	85.74	85.74
	Simple	27.10	77.61	40.18	—	—	—
CSCL	SNoW	86.73	85.44	86.08	90.64	92.31	91.46
	NB	81.93	81.80	81.86	86.10	91.42	88.68
	Simple	62.61	56.26	59.27	—	—	—
ADK99		84.5	88.6	86.5	—	—	—
MPRZ99		88.3	87.9	88.1	91.9	92.2	92.0

Table 4.4: Results of different methods on SV recognition

Muñoz et al. (1999) (MPRZ99), respectively.

The first important observation is that the SV identification task is significantly more difficult than the NP task. This is consistent for all models and all features sets. To confirm this, we look at the statistics of the NP and SV patterns in the training and test corpus as shown in Table 4.5. On average, SV patterns are twice longer than NPs. Moreover, SV patterns occur much less frequently

Data	NP		SV	
	Patterns/Sentence	Length	Patterns/Sentence	Length
Training	6.13	2.17	1.53	4.40
Test	6.13	2.17	1.58	4.42

Table 4.5: Average number of patterns per sentence and average length of patterns of NP and SV patterns in the training and test corpus

than the NPs. These statistics explain the difficulty of the SV over the NP recognition task.

When comparing between different models and features sets, it is clear that the simple HMM formalism is not competitive with the other models. What is interesting here is the very significant sensitivity to the feature base of the classifiers used, despite the violation of the probabilistic assumptions.

For the easier NP task, all models are competitive with one another when the classifiers used are NB or SNoW. Our best results on the NP task are comparable to state-of-the-art systems on this task. All the results listed in the tables above were measured on the same data sets. Note that there are a few other systems that produce comparable $F_{\beta=1}$ measure to the one shown here on a similar, but not identical, task. The system developed by Sha and Pereira (2003) uses a CRF model and achieves 94.38 $F_{\beta=1}$; this is measured, however, on the NP data defined by the CoNLL-2000 text chunking shared task (Tjong Kim Sang, 2002) which is not identical to the base NP data defined by Ramshaw and Marcus (1995) used by all systems shown in table 4.3 including ours. The system by Carreras, Màrquez, and Castro (2005) makes use of an interesting hierarchical approach based on voted Perceptron and achieves even higher $F_{\beta=1}$, at 94.43, on the NP data from the shared task, by training together with all phrase types as defined by CoNLL-2000 shared task.

On the harder, SV, task, CSCL performs better than the other probabilistic methods with HMM not being competitive with others at all. We attribute it to CSCL's ability to cope better with the length of the phrase and the long-term dependencies among the observations. One may improve the classifiers for CSCL as was done by Muñoz et al. (1999). There, the *End* classifier incorporates, in addition to the standard features used here, features extracted from its matching *Beginning*. This

is perhaps the source of the superior performance they had with a similar technique. However, to maintain the fair comparison among different approaches in our experiment, this feature type is discarded.

4.7 Summary

In this chapter, we have developed three inference with classifiers based approaches to the problem of shallow parsing. The first two approaches are within a probabilistic framework that extends standard HMMs in two ways. First, we extended HMMs so that they can take outputs of classifiers and use them as observation probabilities. Second, we modified the assumption of the HMMs and developed a, related, conditional model called PMM, which can also be used to infer global predictions from the outputs of the local classifiers. The third approach is based on an extension of the CSP formalism to handle the probabilistic variables. Outcomes of classifiers are used as input to a CSP, and weighted constraints guide the search for an optimal solution to the inference problem. In both cases, we also provide efficient inference algorithms.

Our systems exhibited state of the art, and very informative results. The results suggest that the CSP formalisms, which supports complex constraints and dependencies flexibly, compares favorably with the probabilistic approaches, which are extensions of HMMs—the standard and commonly used techniques.

We note that approaches related to those studied here have already shown to be successful in practice. Interestingly, both in the context of the chunking problem (Li and Roth, 2001; Zhang, Damerau, and Johnson, 2001) and in the context of named entity recognition (Tjong Kim Sang and Meulder, 2003) the key approach that has been used in the last couple of years is actually the HMM with classifiers—the weakest method we study here. We attribute it to the simplicity of the model and its implementation, rather than to any advantage it has, and hope that the two other models that we have shown superior here will be also studied on more problems. It is also interesting to note that all approaches presented in this chapter can be directly applied to many problems of the

same nature as the shallow parsing problem presented here, i.e. identifying patterns in sequences, as well as in other domains beyond natural language processing. For example, the CSCL approach has been successfully applied to an application in the computational biology (Chuang and Roth, 2001).

Chapter 5

Application II: Semantic Role Labeling

Semantic role labeling (SRL) is the task to identify, for each verb in a sentence, its arguments, such as Agent, Patient or Instrument, as well as its adjuncts, such as Locative, Temporal or Manner. This task is believed to be an important task toward natural language understanding, and has immediate applications in tasks such information extraction and question answering.

In this chapter, we develop two inference with classifiers-based SRL systems. Both systems are trained and evaluated on the PropBank data (Kingsbury and Palmer, 2002) which is a large human-annotated corpus of semantic verb-argument relations. Specifically we use the data pre-processed by the CoNLL-2004 and CoNLL-2005 shared tasks of semantic-role labeling (Carreras and Màrquez, 2004; Carreras and Màrquez, 2005), and follow strictly the task definitions provided there.

The first SRL system is developed for the CoNLL-2004 shared task dataset. For this dataset, the SRL system takes as input only partial syntactic information without any external lexico-semantic knowledge bases. Precisely, the input resources include only a part-of-speech tagger, a shallow parser that can process the input to the level of based chunks and clauses (Tjong Kim Sang and Buchholz, 2000; Tjong Kim Sang and Déjean, 2001), and a named-entity recognizer (Tjong Kim Sang and De Meulder, 2003). We do *not* assume a full parse as input for this dataset. On the other hand, the CoNLL-2005 shared task dataset, which is used to develop the second SRL system, contains the additional full parse tree information with larger amount of training examples.

SRL is a difficult task, and one cannot expect high levels of performance from either purely manual classifiers or purely learned classifiers. Rather, supplemental linguistic information must be used to support and correct a learning system. So far, machine learning approaches to SRL have

incorporated linguistic information only implicitly, via the classifiers' features. The key innovation in our approach is the development of a principled method to combine machine learning techniques with linguistic and structural constraints by explicitly incorporating inference into the decision process.

At the classification level, both systems we present here are composed of two main phases. First, a set of argument candidates is produced by learned classifier(s). In a second learning phase, the candidate arguments from the first phase are re-scored using a classifier designed to determine the argument type for any given candidate argument.

At the inference level, global properties of the sentence, which is difficult to be incorporated into the learning stage, are used to infer the final outputs. These properties are present in the form of structural and linguistic constraints which restrict the set of possible outputs. Sample constraints include "arguments cannot structurally overlap", or, given a predicate, some argument structures are illegal, . The constraints are encoded as linear (in)equalities, and the integer linear programming(ILP) is used as an inference procedure to make a final decision that is both consistent with the constraints and most likely according to the learning system.

Although ILP is generally a computationally hard problem, there are efficient implementations that can run on thousands of variables and constraints. In our experiments, we used the commercial ILP package (Dash Optimization, 2004),

In the rest of the chapter, we start by introducing the definition of the semantic role labeling task in Section 5.1. Section 5.2 presents our first SRL system. Experimental results on CoNLL-2004 dataset are presented in 5.3. Section 5.4, then, presents our second SRL system that makes use of full parse tree information. The experimental results on CoNLL-2005 dataset by this system is then presented in 5.5.

5.1 Semantic Role Labeling Task Definition

The goal of the semantic-role labeling task is to discover the verb-argument structure for a given input sentence. For example, given a sentence:

I *left* my pearls to my daughter-in-law in my will.

the goal is to identify different arguments of the verb *left* which yields the output:

[A0 I] [V *left*] [A1 my pearls] [A2 to my daughter-in-law] [AM-LOC in my will].

Here A0 represents the *leaver*, A1 represents the *thing left*, A2 represents the *benefactor*, AM-LOC is an adjunct indicating the location of the action, and V determines the verb. In addition, each argument can be mapped to a constituent in its corresponding syntactic full parse tree.

Following the definition of the PropBank and CoNLL-2004 shared task, there are six different types of arguments labeled as A0-A5 and AA. These labels have different semantics for each verb as specified in the PropBank Frame files. In addition, there are also 13 types of adjuncts labeled as AM-*adj* where *adj* specifies the adjunct type. In some cases, an argument may span over different parts of a sentence, the label C-*arg* is used to specify the continuity of the arguments, as shown in the example below.

[A1 The pearls] , [A0 I] [V *said*] , [C-A1 were left to my daughter-in-law].

Additionally, an argument might be a relative pronoun that in fact refers to the actual agent outside the clause. In this case, the actual agent is labeled as the appropriate argument type, *arg*, while the relative pronoun is instead labeled as R-*arg*. For example,

[A1 The pearls] [R-A1 which] [A0 I] [V *left*] , [A2 to my daughter-in-law] are fake.

See the details of the definition in (Kingsbury and Palmer, 2002) and (Carreras and Màrquez, 2004).

5.2 System Architecture I: With Partial Parsing Information

The main assumption in our first semantic role labeling system is the availability of only partial parsing information—full parsing information is not present. Following the inference with classifier framework, we split the system into two levels—classification and inference. Moreover, we separate the classification level into two stages. The first stage finds a subset of arguments from all possible candidates. The goal here is to filter out as many as possible false argument candidates, while still maintaining high recall. The second stage focuses on identifying the types of those argument candidates. Since the number of candidates in this stage is much fewer that the first, the second stage is able to use slightly more complicated features to facilitate learning a better classifier. In both learning stages, the learning algorithm used is SNoW, a multiclass classifier that is specifically tailored for large scale learning tasks (see Section 2.1.2). At the end, linguistic and structural constraints are incorporated in the inference level to resolve inconsistent global predictions.

This section describes how we learn the classifiers in both stages, and explain how inference is done in the inference level.

5.2.1 Argument Identification

The first stage is to predict the argument candidates of a given sentence that correspond to the active verb. Since it is difficult to predict the exact arguments accurately, the goal here is to output a superset of the correct arguments by filtering out unlikely candidates.

Specifically, we learn two classifiers, one to detect beginning argument locations and the other to detect end argument locations. Each multiclass classifier makes predictions over forty-three classes—thirty-two argument types, ten continuous argument types, and one class to detect *not beginning/not end*. Features used for these classifiers are:

- **Word** feature includes the current word, two words before and two words after.

- **Part-of-speech tag** (POS) feature includes the POS tags of all words in a window of size two.
- **Predicate lemma & POS tag** show the lemma form and POS tag of the active predicate.
- **Voice** feature is the voice (active/passive) of the current predicate. This is extracted with a simple rule: a verb is identified as passive if it follows a to-be verb in the same phrase chunk and its POS tag is VBN(past participle) or it immediately follows a noun phrase.
- **Position** feature describes if the current word is before or after the predicate.
- **Chunk tag** feature includes the BIO tags for chunks of all words in a window of size two.
- **Chunk pattern** encodes the sequence of chunks from the current words to the predicate.
- **Clause tag** indicates the boundary of clauses.
- **Clause path** feature is a path formed from a semi-parsed tree containing only clauses and chunks. Each clause is named with the chunk preceding it. The clause path is the path from predicate to target word in the pseudo-parse tree.
- **Clause position** feature is the position of the target word relative to the predicate in the pseudo-parse tree containing only clauses. There are four configurations—target word and predicate share the same parent, target word parent is an ancestor of predicate, predicate parent is an ancestor of target word, or otherwise.

Because each argument consists of a single beginning and a single ending, these classifiers can be used to construct a set of potential arguments (by combining each predicted *begin* with each predicted *end* after it of the same type).

Although this stage identifies typed arguments (i.e. labeled with argument types), the second stage will re-score each phrase using phrase-based classifiers—therefore, the goal of the first phase is simply to identify non-typed phrase candidates. In this task, we achieves 98.96% and 88.65% recall (overall, without verb) on the training and the development set, respectively. Because these

are the only candidates passed to the second stage, the final system performance is upper-bounded by 88.65%.

5.2.2 Argument Classification

The second stage in the classification level assigns the final argument classes to (a subset) of the argument candidates supplied from the first stage. Again, the SNoW learning architecture is used to train a multiclass classifier to label each argument to one of the argument types, plus a special class—*no argument (null)*. Training examples are created from the argument candidates supplied from the first phase using the following features:

- **Predicate lemma & POS tag, voice, position, clause Path, clause position, chunk pattern** Same features as those in the first phase.
- **Word & POS tags** from the argument, including the first,last,and head[1] word and tag.
- **Boundary words & POS tag** include two words/tags before and after the target argument.
- **Bigrams** are pairs of words/tags in the window from two words before the target to the first word of the target, and also from the last word to two words after the argument.
- **Phrase type** uses simple heuristics to identify the target argument as VP, PP, or NP.
- **Named entity** feature tells if the target argument is, embeds, overlaps, or is embedded in a named entity with its type.
- **Pseudo-subcategorization** describes the phrase structure around the predicate. We separate the clause where the predicate is in into three parts—the predicate chunk, segments before and after the predicate, and use the sequence of phrase types of these three segments.
- **Verb class** feature is the class of the active predicate described in PropBank Frames.
- **Lengths** of the target argument, in the numbers of words and chunks separately.

[1]We use simple rules to first decide if a candidate phrase type is VP, NP, or PP. The headword of an NP phrase is the right-most noun. Similarly, the left-most verb/proposition of a VP/PP phrase is extracted as the headword

- **Chunk** tells if the target argument is, embeds, overlaps, or is embedded in a chunk with its type.
- **Chunk pattern length** feature counts the number of patterns in the argument.
- **Clause coverage** describes how much of the local clause (from the predicate) is covered by the target argument. It is round to the multiples of 1/4.
- **NEG** feature is active if the target verb chunk has `not` or `n't`.
- **MOD** feature is active when there is a modal verb in the verb chunk. The rules of the **NEG** and **MOD** features are used in a baseline SRL system developed by Erik Tjong Kim Sang (Carreras and Màrquez, 2004).

Although the predictions of the classifier in the second stage can be used directly, the labels of arguments in a sentence often violate some constraints. Therefore, we rely on the inference procedure to make the final predictions.

5.2.3 Inference

The purpose of the inference is to incorporate some prior linguistic and structural knowledge, such as "arguments do not overlap" or "each verb takes at most one argument of each type." This knowledge is used to resolve any inconsistencies of argument classification in order to generate final legitimate predictions. We design an inference procedure based on integer linear programming (ILP). It takes as input the confidence scores over each type of the arguments supplied by the argument classifier. The output is the optimal solution that maximizes the linear sum of the confidence scores (e.g., the conditional probabilities estimated by the argument classifier), subject to the constraints that encode the domain knowledge.

In this subsection we first introduce the constraints and the inference problem in the semantic role labeling task. We then demonstrate how we apply ILP to generate the global label assignment.

Constraints over Argument Labeling

Formally, the argument classifier attempts to assign labels to a set of arguments, $S^{1:M}$, indexed from 1 to M. Each argument S^i can take any label from a set of argument labels, $\mathcal{P}$, and the indexed set of arguments can take a set of labels, $c^{1:M} \in \mathcal{P}^M$. If we assume that the classifier returns a score, $f(S^i = c^i)$, corresponding to the likelihood of seeing label c^i for argument S^i, then, given a sentence, the unaltered inference task is solved by maximizing the overall score of the arguments,

$$\widehat{c}^{1:M} = \underset{c^{1:M} \in \mathcal{P}^M}{\arg\max} f(S^{1:M} = c^{1:M}) = \underset{c^{1:M} \in \mathcal{P}^M}{\arg\max} \sum_{i=1}^{M} f(S^i = c^i). \tag{5.1}$$

In the presence of global constraints derived from linguistic information and structural considerations, our system seeks for a *legitimate* labeling that maximizes the score. Specifically, it can be viewed as the solution space is limited through the use of a filter function, $\mathcal{F}$, that eliminates many argument labelings from consideration. It is interesting to contrast this with previous work that filters individual phrases (Carreras, Màrquez, and Castro, 2005). Here, we are concerned with global constraints as well as constraints on the arguments. Therefore, the final labeling becomes

$$\widehat{c}^{1:M} = \underset{c^{1:M} \in \mathcal{F}(\mathcal{P}^M)}{\arg\max} \sum_{i=1}^{M} f(S^i = c^i) \tag{5.2}$$

The filter function used considers the following constraints:

1. Arguments cannot cover the predicate except those that contain only the verb or the verb and the following word.
2. Arguments cannot overlap with the clauses (they can be embedded in one another).
3. If a predicate is outside a clause, its arguments cannot be embedded in that clause.
4. No overlapping or embedding arguments.
5. No duplicate argument classes for core arguments, such as A0–A5 and AA.

6. Exactly one V argument per proposition (i.e., a sentence given the active verb).
7. If there is a C-V argument, then there should be a sequence of consecutive V, A1, and C-V pattern. For example, when *split* is the verb in "split it up", the A1 argument is "it" and C-V argument is "up".
8. If there is an R-*arg* argument, then there has to be an *arg* argument. That is, if an argument is a reference to some other argument *arg*, then this referenced argument must exist in the sentence.
9. If there is a C-*arg* argument, then there has to be an *arg* argument; in addition, the C-*arg* argument must occur after *arg*. This is stricter than the previous rule because the order of appearance also needs to be considered.
10. Given the predicate, some argument classes are illegal (e.g. predicate 'stalk' can take only A0 or A1). This linguistic information can be found in *PropBank Frames*.

We reformulate the constraints as linear (in)equalities by introducing indicator variables. The optimization problem (Equation 5.2) is solved using ILP.

Using Integer Linear Programming

As discussed previously, a collection of potential arguments is not necessarily a valid semantic labeling since it must satisfy all of the constraints. In this context, inference is the process of finding the *best* (according to Equation 5.1) valid semantic labels that satisfy all of the specified constraints. We take a similar approach that has been previously used for entity/relation recognition (Roth and Yih, 2002), and model this inference procedure as solving an ILP problem.

To solve the problem of Equation 5.2 in this setting, we first reformulate the original cost function $\sum_{i=1}^{M} f(S^i = c^i)$ as a linear function over several binary variables, and then represent the filter function $\mathcal{F}$ using linear inequalities and equalities.

We set up a bijection from the semantic labeling to the variable set $\mathbf{z}$. This is done by setting $\mathbf{z}$ to a set of indicator variables. Specifically, let $z_{ic} = [S^i = c]$ be the indicator variable that represents

whether or not the argument type c is assigned to S^i, and let $p_{ic} = f(S^i = c)$. Equation 5.1 can then be written as an ILP cost function as

$$\arg\max_{\mathbf{z}\in\{0,1\}^d} \sum_{i=1}^{M} \sum_{c=1}^{|\mathcal{P}|} p_{ic} z_{ic},$$

subject to

$$\sum_{c=1}^{|\mathcal{P}|} z_{ic} = 1 \qquad \forall z_{ic} \in \mathbf{z},$$

which means that each argument can take only one type. Note that this new constraint comes from the variable transformation, and is not one of the constraints used in the filter function $\mathcal{F}$.

Constraints 1 through 3 can be evaluated on a per-argument basis—for the sake of efficiency, arguments that violate these constraints are eliminated even before given to the argument classifier. Next, we show how to transform the constraints in the filter function into the form of linear (in)equalities over $\mathbf{z}$, and use them in this ILP setting.

Constraint 4: No overlapping or embedding If arguments $S^{j_1}, \ldots, S^{j_k}$ occupy the same word in a sentence, then this constraint restricts only one of the arguments to be assigned to an argument type. In other words, $k-1$ arguments will be the special class *null*, which means the argument candidate is not a legitimate argument. If the special class *null* is represented by the symbol ϕ, then for every set of such arguments, the following linear equality represents this constraint.

$$\sum_{i=1}^{k} z_{j_i \phi} = k - 1$$

Constraint 5: No duplicate argument classes Within the same sentence, several types of arguments cannot appear more than once. For example, a predicate can only take one A0. This constraint can be represented using the following inequality.

$$\sum_{i=1}^{M} z_{i\text{A0}} \leq 1$$

Constraint 6: Exactly one V argument For each verb, there is one and has to be one V argument, which represents the active verb. Similarly, this constraint can be represented by the following equality.

$$\sum_{i=1}^{M} z_{i\text{V}} = 1$$

Constraint 7: V–A1–C-V pattern This constraint is only useful when there are three consecutive candidate arguments in a sentence. Suppose arguments $S^{j_1}, S^{j_2}, S^{j_3}$ are consecutive. If S^{j_3} is C-V, then S^{j_1} and S^{j_2} have to be V and A1, respectively. This if-then constraint can be represented by the following two linear inequalities.

$$z_{j_3\text{C-V}} \leq z_{j_1\text{V}}, \text{ and } z_{j_3\text{C-V}} \leq z_{j_2\text{A1}}$$

Constraint 8: R-*arg* arguments Suppose the referenced argument type is A0 and the referential type is R-A0. The linear inequalities that represent this constraint are:

$$\forall m \in \{1, \ldots, M\} : \sum_{i=1}^{M} z_{i\text{A0}} \geq z_{m\text{R-A0}}$$

If there are γ reference argument pairs, then the total number of inequalities needed is γM.

Constraint 9: C-*arg* arguments This constraint is similar to the reference argument constraints. The difference is that the continued argument *arg* has to occur before C-*arg*. Assume that the argument pair is A0 and C-A0, and argument S_{j_i} appears before S_{j_k} if $i \leq k$. The linear inequalities that represent this constraint are:

$$\forall m \in \{2, \ldots, M\} : \sum_{i=1}^{m-1} z_{j_i\text{A0}} \geq z_{m\text{C-A0}}$$

Constraint 10: Illegal argument types Given a specific verb, some argument types should never occur. For example, most verbs do not have arguments A5. This constraint is represented by

summing all the corresponding indicator variables to be 0.

$$\sum_{i=1}^{M} z_{i\text{A5}} = 0$$

Using ILP to solve this inference problem enjoys several advantages. Linear constraints are very general, and are able to represent many types of constraints. Previous approaches usually rely on dynamic programming to resolve non overlapping/embedding constraints (i.e., Constraint 4) when the constraint structure is sequential, but are unable to handle other constraints. The ILP approach is flexible enough to handle more expressive constraints. Although solving an ILP problem is NP-hard, with the help of today's commercial numerical packages, this problem can usually be solved very fast in practice. Note that ordinary search methods (e.g., beam search) are not necessarily faster than solving an ILP problem and do not guarantee the optimal solution.

5.3 Experiment I: CoNLL-2004 Dataset

The system is evaluated on the dataset provided in the CoNLL-2004 shared tasks. The dataset consists of a portion of PropBank corpus (Kingsbury and Palmer, 2002) which is the additional annotation of predicate-argument structure over the existing TreeBank (Marcus, Santorini, and Marcinkiewicz, 1993). The training set is extracted from TreeBank section 15–18, the development set, used in tuning parameters of the system, from section 20, and the test set from section 21.

Similar to the shallow parsing problem presented in previous chapter, the performance of the system is measured in terms of $F_{\beta=1}$ defined by

$$F_\beta = \frac{(\beta^2+1)\cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

Recall is the percentage of correct phrases that are identified, and *precision* is the percentage of identified phrases that are indeed correct.

	Prec.	Rec.	$F_{\beta=1}$
1^{st}-phase, non-overlap	70.54	61.50	65.71
1^{st}-phase, All Const.	70.97	60.74	65.46
2^{nd}-phase, non-overlap	69.69	64.75	67.13
2^{nd}-phase, All Const.	71.96	64.93	68.26

Table 5.1: Summary of experiments on the development set. All results are for overall performance.

Table 5.1 summarizes the overall performance of the system on the development set ($2^{nd} - stage, AllConstraints$). In addition the table also shows how additional constraints over the standard non-overlapping constraints improve performance. Moreover, since the argument scoring can be chosen directly from the first stage, this is also evaluated by considering simply the non-overlapping/embedding constraint or the full set of linguistic constraints. The results show the benefit of rescoring by the second stage. Note that by using directly the score from the first stage with only the non-overlapping/embeddning constraint, the system is equivalent to the CSCL presented in Section 4.5.

In general, using all constraints increases $F_{\beta=1}$ by about 1 point in this system, but slightly decreases the performance when only the first stage classifier is used. Also, using the two-stage architecture improves both precision and recall, and the enhancement reflected in $F_{\beta=1}$ is about 2.5 points.

It is interesting to find out how well the classifier in the second stage can perform given perfectly segmented arguments. This evaluates the quality of the argument classifier, and also provides a conceptual upper bound. Table 5.2 first shows the results without using inference (i.e. $\mathcal{F}(\mathcal{P}^M) = \mathcal{P}^M$). The second row shows adding inference to the phrase classification can further improve $F_{\beta=1}$ by 1 point. Note that in this case, the non-overlapping/embedding is not necessary; therefore, only other linguistic constraints are used in the inference. This further illustrates the benefit of the linguistic constraints.

Finally, the overall result on the test set is given in Table 5.3. Our system was the second best at CoNLL-04 shared task, where the best system (Hacioglu et al., 2004) achieve a 69.49 $F_{\beta=1}$ score.

	Precision	Recall	$F_{\beta=1}$
Without Inference	86.95	87.24	87.10
With Inference	88.03	88.23	88.13

Table 5.2: Results of second stage phrase prediction and inference assuming *perfect boundary detection* in the first stage on the development set

	Prec.	Rec.	$F_{\beta=1}$
Overall	70.07	63.07	66.39
A0	81.13	77.70	79.38
A1	74.21	63.02	68.16
A2	54.16	41.04	46.69
A3	47.06	26.67	34.04
A4	71.43	60.00	65.22
AM-ADV	39.36	36.16	37.69
AM-CAU	45.95	34.69	39.53
AM-DIR	42.50	34.00	37.78
AM-DIS	52.00	67.14	58.61
AM-EXT	46.67	50.00	48.28
AM-LOC	33.47	34.65	34.05
AM-MNR	45.19	36.86	40.60
AM-MOD	92.49	94.96	93.70
AM-NEG	85.92	96.06	90.71
AM-PNC	32.79	23.53	27.40
AM-TMP	59.77	56.89	58.30
R-A0	81.33	76.73	78.96
R-A1	58.82	57.14	57.97
R-A2	100.00	22.22	36.36
R-AM-TMP	54.55	42.86	48.00

Table 5.3: Breakdown results on the test set

5.4 System Architecture II: With Full Parsing Information

Our second SRL system mostly retains the similar overall architecture as in the previous system which consists of two classification stages—*argument identification* and *argument classification*—and the *inference* stage. In addition to these stages, this system employ a *pruning* stage at the beginning to help reduce the number of argument candidates to be considered by the argument identification stage. The learning architecture used is also SNoW. The main distinction of this system from the previous one is in its design to exploit the full parsing information which is not

available in the previous system. This section describes how we build these four-stage architecture, and explain how different features are used in training the classifiers. The inference is similar to that of the previous system which is explained in Section 5.2.3. We will only describe how the inference is extended to combine the outputs from multiple SRL systems. This leads to a significant performance improvement as shown in the experiment later on.

5.4.1 Pruning

Since the boundaries of arguments almost always coincide with those of constituents in the full parse tree of a sentence. We only consider the constituents in the parse tree as argument candidates. This is indeed the main benefit of having parse tree information (Punyakanok, Roth, and Yih, 2005). In this stage, our system exploits the heuristic rules introduced by Xue and Palmer (2004) to filter out simple constituents that are very unlikely to be arguments. The heuristic is a recursive process starting from the verb of which arguments to be identified. It first returns the siblings of the verb as candidates; then it moves to the parent of the verb, and collects the siblings again. The process goes on until it reaches the root. In addition, if a constituent is a PP (propositional phrase), its children are also collected. Candidates consisting of only a single punctuation mark are not considered.

This heuristic works well with the correct parse trees. However, one of the errors by automatic parsers is due to incorrect PP attachment leading to missing arguments. To attempt to fix this, we consider as arguments the combination of any consecutive NP and PP, and the split of NP and PP inside the NP that was chosen by the previous heuristics.

5.4.2 Argument Identification

The argument identification stage utilizes binary classification to identify whether a candidate is an argument or not. We train and apply the binary classifiers on the constituents supplied by the pruning stage.

Many of the features used here are similar to those in the previous architecture. Here we summarize the similarity and the difference. Below is the list of features that are similar to those explained in the previous architecture.

- **Predicate lemma and POS tag**
- **Voice**
- **Phrase type**
- **Position**
- **Verb class**
- **Lengths**
- **Chunk**
- **Chunk pattern**
- **Chunk pattern length**
- **Clause relative position**
- **Clause coverage**

Besides the similar features listed above, we introduce new features that which are extracted from the full parse trees. These features are listed below.

- **Head word and POS tag of the head word** feature provides the head word and its POS tag of the constituent. We use rules introduced by Collins (1999) to extract this feature.
- **First and last words and POS tags** of the constituent.
- **Two POS tags before and after** the constituent.

- **Path** records the traversal path in the parse tree from the predicate to the constituent.
- **Subcategorization** feature describes the phrase structure around the predicate's parent. It records the immediate structure in the parse tree that expands to its parent.

5.4.3 Argument Classification

This stage assigns the final argument labels to the argument candidates supplied from the previous stage. A multiclass classifier is trained to classify the types of the argument candidates. In addition, to reduce the excessive candidates mistakenly output by the previous stage, the classifier can also classify the argument as *null* (meaning "not an argument") to discard the argument.

The features used here are similar to those used in the argument identification stage with the addition of

- **Syntactic frame** describes the sequential pattern of the noun phrases and the predicate in the sentence. This is the feature introduced by Xue and Palmer (2004).
- **Propositional phrase head** is the head of the first phrase after the preposition inside PP.
- **NEG and MOD** as described in Section 5.2.2.
- **Named entities** as described in Section 5.2.2.

5.4.4 Inference

As in the previous system, the purpose of this stage is to incorporate some prior linguistic and structural knowledge in order to resolve any inconsistencies of argument classification. This system uses exactly the same inference procedure as described in Section 5.2.3

5.4.5 Inference with Multiple SRL Systems

The inference process allows a natural way to combine the outputs from multiple argument classifiers. Specifically, given k argument classifiers which perform classification on k argument sets,

$\{S_1, \ldots, S_k\}$. The inference process aims to optimize the objective function:

$$\widehat{c}^{1:N} = \arg\max_{c^{1:N} \in \mathcal{P}^N} \sum_{i=1}^{N} Prob(S^i = c^i),$$

where $S^{1:N} = \bigcup_{i=1}^{k} S_i$, and

$$Prob(S^i = c^i) = \frac{1}{k} \sum_{j=1}^{k} Prob_j(S^i = c^i),$$

where $Prob_j$ is the probability output by system j.

Note that all systems may not output with the same set of argument candidates due to the pruning and argument identification. For the systems that do not output for any candidate, we assign the probability with a prior to this *phantom* candidate. In particular, the probability of the *NULL* class is set to be 0.6 based on empirical tests, and the probabilities of the other classes are set proportionally to their occurrence frequencies in the training data.

For example, Figure 5.1 shows the two candidate sets for a fragment of a sentence, "*..., traders say, unable to* ***cool*** *the selling panic in both stocks and futures.*" In this example, system A has two argument candidates, a_1 = "traders" and a_4 = "the selling panic in both stocks and futures"; system B has three argument candidates, b_1 = "traders", b_2 = "the selling panic", and b_3 = "in both stocks and futures". The phantom candidates are created for a_2, a_3, and b_4 of which probability is set to the prior.

Specifically for this implementation, we first train two SRL systems that use Collins (1999)'s parser and Charniak (2001)'s parser respectively. In fact, these two parsers have noticeably different output. In evaluation, we run the system that was trained with Charniak's parser 5 times with the top-5 parse trees output by Charniak's parser[2]. Together we have six different outputs per predicate. Per each parse tree output, we run the first three stages, namely pruning, argument identification, and argument classification. Then a joint inference stage is used to resolve the

[2]The top parse tree were from the official output by CoNLL. The 2nd-5th parse trees were output by Charniak's parser.

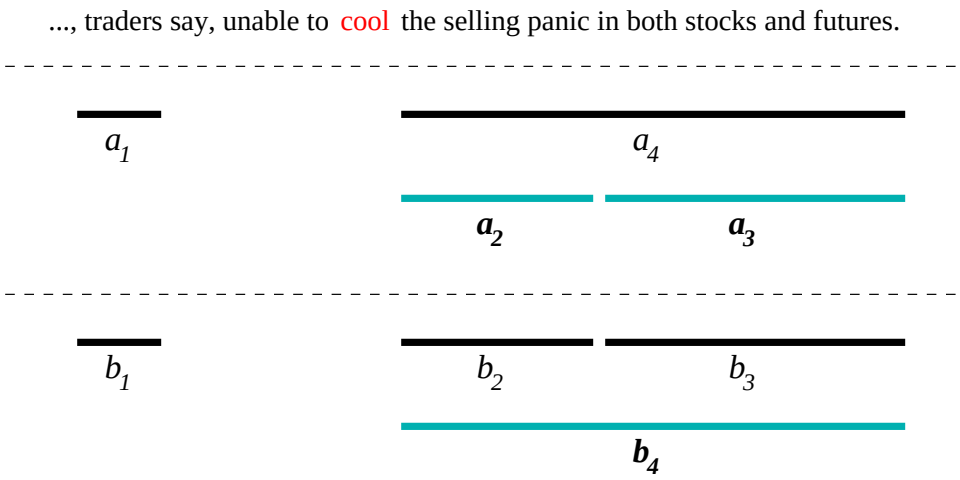

Figure 5.1: Two SRL systems' output (a_1, a_4, b_1, b_2, and b_3), and phantom candidates (a_2, a_3, and b_4).

inconsistency of the output of argument classification in these systems.

5.5 Experiments II: CoNLL-2005 Dataset

The system is evaluated on the dataset provided in the CoNLL-2005 shared tasks. The dataset consists of a portion of the official PropBank I corpus with an additional annotation of predicate-argument structure a portion of Brown corpus. The training set is extracted from section 02–21, the development set, used in tuning parameters of the system, from section 24, and the test set from section 23. In addition, the test set includes three sections of Brown corpus (ck01–03).

	Precision	Recall	$F_{\beta=1}$
Charniak-1	75.40	74.13	74.76
Charniak-2	74.21	73.06	73.63
Charniak-3	73.52	72.31	72.91
Charniak-4	74.29	72.92	73.60
Charniak-5	72.57	71.40	71.98
Collins	73.89	70.11	71.95
Joint inference	80.05	74.83	77.35

Table 5.4: The results of individual systems and the result with joint inference on the development set

	Precision	Recall	$F_{\beta=1}$
Development	80.05	74.83	77.35
Test WSJ	82.28	76.78	79.44
Test Brown	73.38	62.93	67.75
Test WSJ+Brown	81.18	74.92	77.92

Table 5.5: Overall results on test data

Table 5.4 shows the overall performance on the development set of the SRL system using different parse trees outputs, and the result after their outputs are combined by the inference. Combining multiple SRL systems significantly improves the result by 2.65 $F_{\beta=1}$ score. Overall results on the development and test data are shown in Table 5.5. This system was the top system in the CoNLL-2005 shared task. The breakdown of the performance on each argument type on the WSJ test set is shown in Table 5.6.

5.6 Summary

In this chapter we develop two systems to approach the problem of semantic role labeling. Both systems consist of classifiers and a final inference procedure based on integer linear programming to infer the final outputs. The main distinction of the first system from the other is in the use of only partial parsing information, not the full parsing information.

In the first system, the classification level are split into two stages. First, two classifiers are used to identify the beginnings and ends of arguments and the outputs are then combined to form argument candidates. After that, each argument candidate are scored by the argument classifier for each possible argument type. The inference procedure, then, infer the final decision. Linguistic and structural information are encoded as constraints for the inference procedure which are also shown to improve the overall performance of the both systems. This system are applied to the CoNLL-2004 shared task dataset. The performance of the system ranked among the top systems participating in the shared task.

The second system improves over the first by utilizing the full parsing information. The use

Test WSJ	Precision	Recall	$F_{\beta=1}$
Overall	82.28	76.78	79.44
A0	88.22	87.88	88.05
A1	82.25	77.69	79.91
A2	78.27	60.36	68.16
A3	82.73	52.60	64.31
A4	83.91	71.57	77.25
A5	0.00	0.00	0.00
AM-ADV	63.82	56.13	59.73
AM-CAU	64.15	46.58	53.97
AM-DIR	57.89	38.82	46.48
AM-DIS	75.44	80.62	77.95
AM-EXT	68.18	46.88	55.56
AM-LOC	66.67	55.10	60.33
AM-MNR	66.79	53.20	59.22
AM-MOD	96.11	98.73	97.40
AM-NEG	97.40	97.83	97.61
AM-PNC	60.00	36.52	45.41
AM-PRD	0.00	0.00	0.00
AM-REC	0.00	0.00	0.00
AM-TMP	78.16	76.72	77.44
R-A0	89.72	85.71	87.67
R-A1	70.00	76.28	73.01
R-A2	85.71	37.50	52.17
R-A3	0.00	0.00	0.00
R-A4	0.00	0.00	0.00
R-AM-ADV	0.00	0.00	0.00
R-AM-CAU	0.00	0.00	0.00
R-AM-EXT	0.00	0.00	0.00
R-AM-LOC	85.71	57.14	68.57
R-AM-MNR	0.00	0.00	0.00
R-AM-TMP	72.34	65.38	68.69

Table 5.6: Breakdown of the performance on WSJ test set

of full parsing information allows the system to consider only the constituents in the parse trees instead of all possible chunks in the sentences. At the classification level, the system employs simple heuristic rules to prune out obvious non-argument constituents. Constituents are then identified as arguments by an argument identifier which are, in turn, scored for each argument type by an argument classifier. Again, the outputs of the argument classifiers are, then, used by the inference

procedure to infer the final decision satisfying the linguistic and structural constraints. In addition, the inference provides a natural way to take the output of multiple argument classifiers and combines them into a coherent predicate-argument output. Significant improvement in overall SRL performance through this inference is also illustrated. The system are applied to the CoNLL-2005 shared task dataset resulting in the top system among those participating in the shared task.

Chapter 6

Utility of Constraints in Inference

In principle, constraints guarantee the global coherency of the outputs, and, hence, help improve the global performance of a system. However, in real world applications, performance is usually measured in terms of local accuracy or some functions that are more directly influenced by local predictions, e.g. $F_{\beta=1}$ measure. Hence, enforcing global constraints does not guarantee to improve the performance with respect to these performance metrics. Worse, when local classifiers are trained independently to optimize the local accuracy, global constraints should only degrade the performance. Nonetheless, in practice it has been consistently shown that global constraints improve the performance (Roth and Yih, 2005).

Our study aims to develop a better understanding on this issue. Our investigation, however, is restricted to only the case of *non-interactive* classifiers with linear summation scoring function in the inference. In spite of the simplicity of this formulation, it has been shown to perform very well in real world problems. Two very obvious examples are the CSCL formulation for the shallow parsing task presented in Chapter 4 and the systems presented in Chapter 5 for the semantic role labeling task. Specifically, we show that, with respect to Hamming loss, the performance of a system may degrade with the use of constraints, and develop a sufficient condition to guarantee that using constraints cannot hurt the system. In addition, we presents an empirical study on a real world system, i.e. the SRL system II presented in Chapter 5, which leads to some explanations why constraints do not impair the system even if the sufficient condition is not fully satisfied.

We start the chapter by rigorously formulating the problem that we are interested in. Then, in Section 6.2 we develop some theoretical guarantees when constraints do cannot impair the system performance. Finally, the further empirical study on a real world system is presented in Section 6.3.

6.1 Problem

Given a system h that predicts an output $\mathbf{y} \in \mathcal{Y}$ for any input $\mathbf{x} \in \mathcal{X}$. Without the loss of generality, both $\mathbf{x}$ and $\mathbf{y}$ are assumed to be sequences $\langle x_1, \ldots, x_l \rangle$ and $\langle y_1, \ldots, y_l \rangle$ respectively. Given an example $(\mathbf{x}, \mathbf{y})$ where $\mathbf{y}$ is the correct output that h is expected to produce for $\mathbf{x}$, the quality of the output of h on $\mathbf{x}$, i.e. $h(\mathbf{x})$ is measured in terms of some loss function $\ell(\mathbf{y}, h(\mathbf{x}))$. The overall quality of the system h is then measured in terms of expected loss over all examples:

$$\ell(h) = \mathbf{E}_{\mathbf{x},\mathbf{y}} \ell(\mathbf{y}, h(\mathbf{x})) = \sum_{\mathbf{x} \in \mathcal{X}, \mathbf{y} \in \mathcal{Y}} P(\mathbf{x}, \mathbf{y}) \ell(\mathbf{y}, h(\mathbf{x}))$$

where $\mathbf{E}_{\mathbf{x},\mathbf{y}}$ denotes the expectation with respect to true underlying (possibly unknown) distribution of $(\mathbf{x}, \mathbf{y})$. In machine learning, the goal is usually to find h that minimizes this function. Since the true underlying distribution is usually unknown, the performance of a system is instead measured on a validation set $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$, and, therefore,

$$\ell(h|\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \ell(\mathbf{y}, h(\mathbf{x})).$$

When comparing two systems, h_1 and h_2, we consider h_1 to be better than h_2 if $\ell(h_1) < \ell(h_2)$.

In this chapter, the goal is not to learn h, but to study some properties of *inference with classifiers* systems:

$$h = \arg\max_{\mathbf{y} \in \mathcal{C}(\mathcal{Y})} score(\mathbf{x}, \mathbf{y} | f_1, f_2, \ldots, f_l),$$

where $score$ is some global objective function over $\mathbf{x}$ and $\mathbf{y}$ given a set of classifiers $\{f_i\}$. The aim is to develop some understanding of the usefulness of the constraints $\mathcal{C}(\cdot)$—when and how using $\mathcal{C}(\cdot)$ affect the performance of a system. In this study, we assume that 1) classifiers are *non-interactive*:

$$f_i(\mathbf{x}, \mathbf{y}_{\setminus i}, y_i) \equiv f_i(\mathbf{x}, y_i),$$

and 2) the scoring function is a linear summation:

$$score(\mathbf{x}, \mathbf{y}|f_1, f_2, \ldots, f_l) = \sum_{1}^{l} f_i(\mathbf{x}, y_i).$$

Hence,

$$h = \arg\max_{\mathbf{y} \in \mathcal{C}(\mathcal{Y})} \sum_{i=1}^{l} f_i(\mathbf{x}, y_i).$$

Despite the simplicity of these assumptions, the formulation has been successfully applied in real world problems, e.g. the CSCL formulation for the shallow parsing task presented in Chapter 4 and the systems presented in Chapter 5 for the semantic role labeling task.

Specifically, we compare the following two types of inference.

Unconstrained Inference

$$h_{\text{Un}}(\mathbf{x}) \quad = \quad \arg\max_{\mathbf{y} \in \mathcal{Y}} \sum_{i=1}^{l} f_i(\mathbf{x}, y_i)$$

Note that this is equivalent to each classifier making its prediction independently.

Constrained Inference

$$h_{\text{Con}}(\mathbf{x}) \quad = \quad \arg\max_{\mathbf{y} \in \mathcal{C}(\mathcal{Y})} \sum_{i=1}^{l} f_i(\mathbf{x}, y_i)$$

for some constrained set $\mathcal{C}(\mathcal{Y})$.

In this work, we focus on a local performance measure, i.e. Hamming loss that measures the performance in terms of the number of local mistakes. Formally, the loss function is defined as:

$$\ell_{\text{Ham}}(\mathbf{y}, \mathbf{y}') = \frac{1}{l} \sum_{i=1}^{l} loss(y_i, y_i'),$$

where

$$loss(y, y') = \begin{cases} 0 & y = y', \\ 1 & y \neq y'. \end{cases}$$

This loss closely represents the performance measures in real world applications which are more directly influenced by local decisions regardless of the global quality, e.g. F1 measure.

Since Hamming loss measures the performance directly on the local decisions, it is not hard to see that to achieve the optimal performance, the system should predict for each local decision the output that maximizes the posterior probability. That is,

$$h(\mathbf{x}) = \langle f_1(\mathbf{x}), \ldots, f_l(\mathbf{x}) \rangle$$

where

$$f_i(\mathbf{x}) = \arg\max_{y_i \in \mathcal{Y}_i} P(y_i|\mathbf{x}).$$

This suggest that, at least when the local classifiers can achieve this optimality, enforcing global coherency can only degrade the performance of the system with respect to the Hamming loss. This is contradict to the observation in real word applications that has been consistently shown that global constraints improve the (local) performance (Roth and Yih, 2005). In the next section, we will further investigate this issue and develop a sufficient condition when a system cannot be hurt by global constraints.

In contrast to Hamming loss, the zero-one loss formally defined as:

$$\ell_{0-1}(\mathbf{y}, \mathbf{y}') = \begin{cases} 0 & \mathbf{y} = \mathbf{y}', \\ 1 & \mathbf{y} \neq \mathbf{y}' \end{cases}$$

penalizes the mistake uniformly no matter how different the output is, comparing to the correct output. It is not hard to show that, with respect to this loss, constraints can only improve the systems. We will show this in the next section.

6.2 A Sufficient Condition

In this section, we begin by giving a trivial proof confirming the utility of constraints with respect to zero-one loss. Then, we move on to analyze the Hamming loss, the main focus of this chapter, and give a sufficient condition to guarantee that a system cannot be impaired by using constraints.

First, in case of zero-one loss, it can be easily shown that constraints can never hurt the performance. This is summarized in the following claim.

Claim 6.1. *For any constraint set $\mathcal{C}(\mathbf{y})$, and classifiers f_i, $\ell_{0\text{-}1}(h_{Con}) \leq \ell_{0\text{-}1}(h_{Un})$.*

Proof. For any example $(\mathbf{x}, \mathbf{y}')$ such that h_{Un} is correct, that is, $h_{\text{Un}}(\mathbf{x}) = \mathbf{y}'$, h_{Con} is also correct because $\mathbf{y}' \in \mathcal{C}(\mathcal{Y}) \subseteq \mathcal{Y}$. Therefore, $\ell_{0-1}(\mathbf{y}', h_{\text{Cons}}(\mathbf{x}))$ remains 0 on this example. On the other hand, for any $(\mathbf{x}, \mathbf{y}')$ such that $h_{\text{Un}}(\mathbf{x})$ is incorrect, $\ell_{0\text{-}1}(\mathbf{y}', h_{\text{Un}}(\mathbf{x})) = 1 \geq \ell_{0\text{-}1}(\mathbf{y}', h_{\text{Cons}}(\mathbf{x}))$. Since, $\ell_{0-1}(h'(\mathbf{x}), h_{\text{Cons}}(\mathbf{x})) \leq \ell_{0-1}(h'(\mathbf{x}), h_{\text{Un}}(\mathbf{x}))$ for all examples $(\mathbf{x}, \mathbf{y}')$, $\ell_{0\text{-}1}(h_{\text{Con}}) \leq \ell_{0\text{-}1}(h_{\text{Un}})$. □

Unlike the 0-1 loss, with respect to Hamming loss, constraints can hurt the performance of a system. Consider the example shown in Figure 6.1. In this example, f_1 performs perfectly while f_2 always makes mistakes. Hence, h_{Un} makes mistakes on half of its predictions, i.e. $\ell_{\text{Ham}}(h_{\text{Un}}) = 0.5$. However, given the constraints $\mathcal{C}(\mathcal{Y})$, the outputs of h_{Un} satisfy the constraints except for the last example. Since the confidence of f_1 on the correct output, 1, is less than that of f_2 on the incorrect output 0, the inference changes the output of f_1 which results in a higher Hamming loss for h_{Con}. This example demonstrates that constraints can hurt a system with respect to Hamming loss. This also proves the following claim.

Claim 6.2. *There exist a constraint set $\mathcal{C}(\mathbf{y})$ and classifiers f_i such that $\ell_{Ham}(h_{Con}) > \ell_{Ham}(h_{Un})$*

The example also suggests that the degradation happens when the correct classifier is not very confident in its output while the incorrect classifier outputs high level of confidence. When the combination of the outputs violates constraints, it is more likely that the output of the correct classifier is mistakenly fixed. Based on this observation, the rest of this section focuses on developing a sufficient condition to guarantee that constraints cannot degrade the performance.

$\mathcal{C}(\mathcal{Y}) = \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle\}$

Input		Output		f_1		f_2		h_{Un}		h_{Con}	
x_1	x_2	y_1	y_2	0	1	0	1	y_1	y_2	y_1	y_2
0	0	0	0	0.6	0.4	0.2	0.8	0	1	0	1
0	1	0	1	0.6	0.4	0.8	0.2	0	0	0	0
1	0	1	0	0.4	0.6	0.2	0.8	0	1	0	1
1	1	1	0	0.4	0.6	0.2	0.8	1	1	0	1
							$\ell_{\text{Ham}} =$	0.5		0.625	

Figure 6.1: An example where constraints hurt the system with respect to Hamming loss

Given an example $(\mathbf{x}, \mathbf{y}')$, let $\widehat{\mathbf{y}} = h_{\text{Un}}(\mathbf{x})$ and $\mathbf{y}^* = h_{\text{Cons}}(\mathbf{x})$. Clearly, the contribution to the degradation of the Hamming loss may happen at position i of the output only when h_{Un} does not make mistake ($\widehat{y}_i \neq y'_i$). In other word, if h_{Un} makes mistake at $\widehat{y}_i$, h_{Con} does better at this position if $y^*_i = y'_i$; otherwise, the loss at this position remains unchanged. Moreover, this degradation at i can happen only when there are some mistakes at some other positions by h_{Un}, or, otherwise, h_{Un} makes perfect prediction as does h_{Cons}. This argument proves the following proposition.

Proposition 6.3. *For any example* $(\mathbf{x}, \mathbf{y}')$*, let* $\widehat{\mathbf{y}} = h_{Un}(\mathbf{x})$ *and* $\mathbf{y}^* = h_{Con}(\mathbf{x})$*. For any position* i*, if* $loss(y'_i, y^*_i) > loss(y'_i, \widehat{y}_i)$*, then* $\widehat{y}_i \neq y'_i$ *and* $\exists j \neq i : \widehat{y}_j \neq y'_j$*.*

We can further restrict the conditions that can contribute to this degradation.

For any given set of indices $\mathcal{I}$, let

$$f_{\mathcal{I}}(\mathbf{x}, \mathbf{y}_{\setminus i}) = \sum_{i \in \mathcal{I}} f_i(\mathbf{x}, y_i)$$

Let $\mathcal{L} = \{1, \ldots, l\}$ and $\mathcal{L}_{\setminus i} = \mathcal{L} - \{i\}$. Define

$$\gamma_i = \min_{\tilde{y}_i \in \mathcal{Y}_i - \{y'_i\}} f_i(\mathbf{x}, y'_i) - f_i(\mathbf{x}, \tilde{y}_i)$$

$$\delta_i = \max_{\tilde{\mathbf{y}}_{\setminus i} \in \mathcal{Y}_{\setminus i}} f_{\mathcal{L}_{\setminus i}}(\mathbf{x}, \tilde{\mathbf{y}}_{\setminus i}) - f_{\mathcal{L}_{\setminus i}}(\mathbf{x}, \mathbf{y}'_{\setminus i})$$

γ_i indicates how far the confidence of the correct output y_i is from the other possible outputs

when f_i makes correct prediction at this position. When f_i makes a mistake, $-\gamma_i$ indicates the distance between the confidence of the predicted output and the confidence of the correct output. δ_i indicates how far below the confidence of the correct output $\mathbf{y}'_{\setminus i}$ is from the predicted output $\tilde{\mathbf{y}}_{\setminus i}$.

Theorem 6.4. *For any example $(\mathbf{x}, \mathbf{y}')$, let $\widehat{\mathbf{y}} = h_{Un}(\mathbf{x})$ and $\mathbf{y}^* = h_{Con}(\mathbf{x})$. For any position i, if $loss(y'_i, y^*_i) > loss(y'_i, \widehat{y}_i)$ the following properties are satisfied.*

1. $\widehat{y}_i \neq y_i$.
2. $\exists j \neq i : \widehat{y}_j \neq y_j$.
3. $\gamma_i < \delta_i$.

Proof. Properties 1 and 2 follows from Proposition 6.3. We only need to prove for property 3.

$$
\begin{aligned}
\max_{\tilde{\mathbf{y}} \in \mathcal{C}(\mathcal{Y})} \sum_{i \in \mathcal{L}} f_i(\mathbf{x}, \tilde{y}_i) &= \max_{\tilde{\mathbf{y}} \in \mathcal{C}(\mathcal{Y}) | \tilde{y}_i \neq \mathbf{y}'_i} \sum_{i \in \mathcal{L}} f_i(\mathbf{x}, \tilde{y}_i) && (6.1)\\
&\leq \max_{\tilde{\mathbf{y}} \in \mathcal{Y} | \tilde{y}_i \neq \mathbf{y}'_i} \sum_{i \in \mathcal{L}} f_i(\mathbf{x}, \tilde{y}_i) && (6.2)\\
&= \max_{\tilde{y}_i \in \mathcal{Y}_i - \{y'_i\}} f_i(\mathbf{x}, \tilde{y}_i) + \max_{\tilde{\mathbf{y}}_{\setminus i} \in \mathcal{Y}_{\setminus i}} f_{\mathcal{L}_{\setminus i}}(\mathbf{x}, \tilde{\mathbf{y}}_{\setminus i}) && (6.3)\\
&= \max_{\tilde{y}_i \in \mathcal{Y}_i - \{y'_i\}} f_i(\mathbf{x}, \tilde{y}) - f_i(\mathbf{x}, y'_i) \\
&\quad + \max_{\tilde{\mathbf{y}}_{\setminus i} \in \mathcal{Y}_{\setminus i}} f_{\mathcal{L}_{\setminus i}}(\mathbf{x}, \tilde{\mathbf{y}}_{\mathcal{L}_{\setminus i}}) - f_{\setminus i}(\mathbf{x}, \mathbf{y}_{\setminus i}) \\
&\quad + f_i(\mathbf{x}, y'_i) + f_{\mathcal{L}_{\setminus i}}(\mathbf{x}, \mathbf{y}'_{\setminus i}) && (6.4)\\
&= -\gamma_i + \delta_i + \sum_{i \in \mathcal{L}} f_i(\mathbf{x}, y'_i) && (6.5)\\
&< -\gamma_i + \delta_i + \max_{\tilde{\mathbf{y}} \in \mathcal{C}(\mathcal{Y})} \sum_{i \in \mathcal{L}} f_i(\mathbf{x}, \tilde{y}_i) && (6.6)\\
\gamma_i &< \delta_i && (6.7)
\end{aligned}
$$

(6.1) comes from the fact that h_{Con} makes mistake at position i. (6.6) follows from the fact that $\mathbf{y}' \in \mathcal{C}(\mathcal{Y})$, but $\mathbf{y}'$ is not output by h_{Con}. □

Theorem 6.4 suggests that, for a set of classifiers not to be impaired by constraints, when a classifier is correct, it should allow large margin between the confidence of the correct output and an incorrect one. On the other hand, when the output is incorrect, the confidence of the correct one should not be far worse than that of its predicted output.

Theorem 6.4 also provides a sufficient condition when an inference does not degrade by constraints as summarized in the following corollary.

Corollary 6.5. $\ell_{Ham}(h_{Con}) \leq \ell_{Ham}(h_{Un})$ *if, for all examples and positions, all three properties in Theorem 6.4 do not holds at the same time.*

6.3 An Empirical Investigation

The previous section investigates when constraints hurt the system performance. However, with respect to Hamming loss, Corollary 6.4 only provide a sufficient condition to guarantee that constraints cannot hurt the system. The question, however, remains if this condition really holds in a practical system, and if not, why the system can benefit from constraints.

In this section, we investigate a practical system, namely, the semantic role labeling system II presented in Chapter 5 and its predictions on the CoNLL-2005 WSJ test set. The focus is now only on Hamming loss due to its possible degradation of the system when constraints are incorporated, while for zero-one loss, constraints can only benefit.

First, Table 6.1 shows the performance of this system in terms of local accuracy (inverse of Hamming loss) to confirm the benefit of constraints for this system in terms of Hamming loss.

System	Without Constraints	With Constraints
SRL System II	82.38%	84.08%

Table 6.1: Local accuracies of SRL System II on WSJ test data

To explain why constraints help improve the system, we follow Theorem 6.4 and Corollary 6.5 that suggests a sufficient condition for a system not to degrade by constraints. We attempt to

find out whether the condition holds in this system. Table 6.2, however, suggests otherwise. It summarizes the number of local predictions that satisfy three properties in Theorem 6.4, and hence, these examples can potentially be degraded by using constraints in the inference.

	Number	Percentages
Total Predictions	19822	100.00
Incorrect Predictions	3492	17.62
Correct Predictions	16330	82.38
Predictions Satisfying Properties	1833	9.25

Table 6.2: Summary of the predictions by SRL System II on WSJ test data that satisfy properties in Theorem 6.4 when constraints are not used

Although, the system does not totally follows Theorem 6.4, the theorem may suggest that, for those examples and positions that satisfy properties 1 and 2, the larger the γ_i is relative to δ_i, the less likely the output is mistakenly fixed from correct to incorrect. Hence, if most examples have large γ_i relative to δ_i, the chance of the performance being degraded is lower. Table 6.3 provides the statistics to illustrate this point. It shows the almost monotonic decrease in the numbers of examples when the relative margin (γ_i/δ_i) decreases. In addition, it also shows that the almost monotonic increase in percentages of correct predictions that are mistakenly fixed by using constraints when the relative margin decreases. This evidently supports the claim we made.

γ_i/δ_i	Total Examples	Fixed	%
0	117	13	11.11
0.1	89	13	14.61
0.2	89	12	13.48
0.3	133	14	10.53
0.4	138	8	5.80
0.5	237	8	3.38
0.6	251	6	2.39
0.7	222	8	3.60
0.8	260	9	3.46
0.9	297	6	2.02
Total	1833	97	5.29

Table 6.3: Number of the correct predictions of SRL System II that are mistakenly fixed by constraints

The number above, however, only tells one side of the story. It only explains that the constraints may not hurt the performance by much if the relative margin is large. The question why an incorrect prediction turns correct by constraints still remains. An observation on Theorem 6.4 suggests that, for constraints not to be hurtful, δ_i, which indicates the summation of the differences between the confidences of the predicted labels and those of the correct labels, should be small. We take this further to hypothesize that when a classifier makes a mistake, it should not output the confidence of the correct label that is far below that of the predicted one so that constraints are more likely to fix the mistake.

$-\gamma_i$	Total Examples	Fixed	%
0	589	133	22.58
0.1	465	87	18.71
0.2	387	70	18.09
0.3	314	44	14.01
0.4	286	38	13.29
0.5	256	21	8.20
0.6	266	15	5.64
0.7	280	17	6.07
0.8	349	7	2.01
0.9	300	2	0.67
Total	3492	434	12.43

Table 6.4: Number of incorrect predictions by SRL System II that are correctly fixed by constraints

Consider when a classifier makes a mistake, $-\gamma_i$ measures the distance between the confidence of the correct label and that of the predicted one. Table 6.4 summarizes the number of incorrect predictions being correctly fixed by using constraints. The result shows a higher percentages of the predictions being correctly changed for smaller value of $-\gamma_i$. This results confirms the hypothesis that the confidence of the correct label should not be far below that of the predicted one when a classifier makes a mistake.

We have shown that there are cases that constraints can improve and that constraints can degrade the performance. The overall improvement, however, depends on the ratio of between these two cases. The data in Table 6.3 and Table 6.4 together summarizes the overall number of predic-

tions (434-97 = 337) of the system improved by using constraints.

We aim to hypothesize this overall improvement from a global perspective. Given any constraint, if the output from the unconstrained inference does not satisfy the constraints, the inference needs to search for the next solution that satisfies the constraints with the highest confidence. Hence, when comparing to the optimal solution output by the unconstrained inference, if the confidences of better solutions (in terms of Hamming loss) are not far lower, but the confidence of worse solutions are instead far below, the system is more likely to benefit from constraints.

Figure 6.2 shows the relationship between the Hamming loss and the confidence of solutions of the SRL System II on WSJ test data. Precisely, we consider only the solutions that satisfy constraints, and group them according to the gain (or loss) in Hamming loss comparing to optimal solutions. For each possible gain (or loss), we take the average of the relative confidences between the solutions and the optimal solutions. The relative confidence of a solution is the difference in terms of the confidence between the solution and the optimal one. This data is plotted in the chart shown in Figure 6.2. Consider the solutions that are better (gain is grater than 0) and worse than optimal solutions with the same magnitude, we may observe that those that are better are closer, in terms of confidences, to optimal solutions than those that are worse. This suggests that when constraints are incorporated to the inference, it is more likely that the inference outputs a solution that reduces Hamming loss.

The same effect can also be observed in an inverse view shown in Figure 6.3 comparing the average gain in Hamming loss over the optimal solution for each relative confidence. From this perspective, we can see that the closer a solution is to the optimal one in terms of the score, the better it is, on average, in terms of Hamming loss with those that are close to the optimal solutions having lower Hamming loss than the optimal solution.

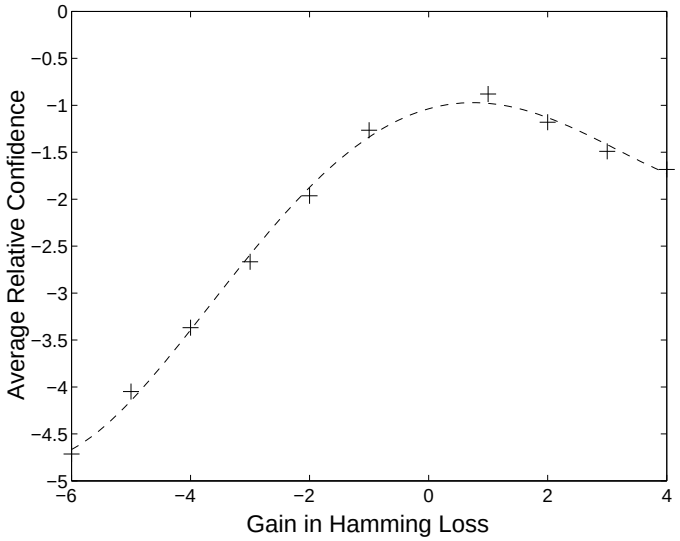

Figure 6.2: Comparison of the average of the relative confidence and the gain in Hamming loss of the solutions that satisfy constraints

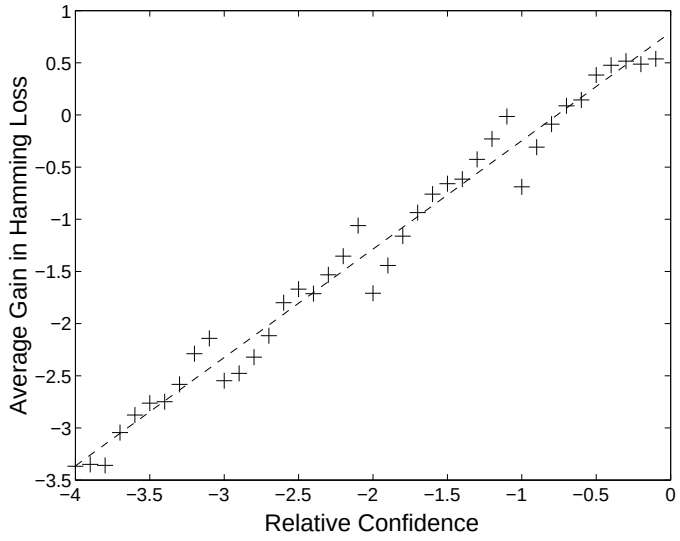

Figure 6.3: Average of gain in Hamming loss per relative confidence of the solutions that satisfy constraints

6.4 Summary

In this chapter, we investigate the usefulness of constraints in the inference with respect to zero-one loss and Hamming loss. We show that with respect to zero-one loss, the constraints can never

degrade the performance of a system. On the other hand, the performance may be degraded by incorporating constraints into the inference if the classifiers do not maintain high margin of separation when it makes a correct prediction, and low margin when it makes a mistake. A sufficient condition for a system not to be degraded by constraints are provided.

Furthermore, we empirically investigate the SRL System II presented in Chapter 5 on the WSJ test data to discuss further why constraints can help; even though the sufficient condition does not hold. We hypothesize that a system that can benefit most from constraints is the one that has good classifiers—maintaining high margin when correct, and low margin when wrong—in most cases. The analysis on the SRL System II supports our claim.

Chapter 7

Conclusions

A very common approach to learning problems with complex and structured output is to decompose the output into components, learning each one, and then using these, along with task and domain specific constraints to infer a coherent output. We provide a unified view for this problem in a framework called *inference with classifiers* in which complex and structured output problems are viewed in two levels: 1) the classification level—where several different classifiers are learned and evaluated to output their confidences over possible predictions—and 2) the inference level—where a final, global, decision is made, based on the outcomes of these classifiers with respect to some domain knowledge encoded in the form of global constraints over the mutual outputs of the local classifiers.

Our framework was applied to two fundamental problems in natural languages processing—shallow parsing and semantic role labeling. In the shallow parsing problem, we develop three inference approaches. The first two are probabilistic approaches that extend standard hidden Markov models (HMM) in two ways. First, we extend the HMMs such that they can take the outputs from classifiers and utilize them in terms of their observation probability. Second, we modify the assumption of the HMMs that results in another Markov model, projection-based Markov model, which can also infer the final predictions from the outputs of classifiers. The last approach is based on an extension of the constraint satisfaction problem (CSP) formalism to handle the probabilistic variables. The outcomes of the classifiers participate in terms of these probabilities and solving this CSP results in the optimal output of the problem. In addition, our approaches developed here may be generally applied to different problems of the similar nature to the shallow parsing problem, that is, to identify some patterns in sequential data. One of our approaches has been successfully

applied to an application in the computational biology (Chuang and Roth, 2001).

For the semantic role labeling (SRL), two systems are developed. The first system uses only partial parsing information while the other system makes use of full parsing information. Both systems formulate the inference as an integer linear program (ILP) allowing structural and linguistic constraints to be incorporated flexibly in the form of linear (in)equalities. In addition, the inference procedure offers a natural extension to combine the outputs from multiple SRL systems which leads to a significant improvement in the overall performance. ILP is a very powerful problem-solving tool, and hence, the inference based on ILP can be very flexibly extended to tackle different problems, not limited to sequential problems.

Although using constraints in the inference have shown to improve the performance in practice (Roth and Yih, 2005), there is no theoretical guarantee that the additional constraints would always help. A rigorous study of the influence of constraints in the inference has never been done. This dissertation takes an initial step in this direction by investigating this issue with respect to two specific loss functions. We show that, with respect to zero-one loss, the performance of a system can be shown not to degrade with the use of constraints while such guarantee does not hold for Hamming loss. In case of Hamming loss, we develop a sufficient condition to guarantee that using constraints cannot hurt the system performance. An empirical study on a real world system leads to some explanations why constraints do not impair the system even if the sufficient condition is not fully satisfied.

In the future, we hope to investigate further the influence of constraints in inference. Even though we have developed a sufficient condition to guarantee that using constraints cannot hurt the system performance, whether the condition may be improved or whether such condition can be developed to guarantee the improvement of the system are still left unanswered. Answers to these problems will lead to a better understanding of the use of constraints, and, hence, a better inference procedure or a better learning algorithm that has its goal to be incorporated in our framework.

Bibliography

Abney, S. P. 1991. Parsing by chunks. In R. C. Berwick, S. P. Abney, and C. Tenny, editors, *Principle-based parsing: Computation and Psycholinguistics*. Kluwer, Dordrecht, pages 257–278.

Agarwal, S. and D. Roth. 2002. Learning a sparse representation for object detection. In *ECCV-2002, The Proceedings of the 8th European Conference on Computer Vision*, pages 113–128.

Appelt, D., J. Hobbs, J. Bear, D. Israel, and M. Tyson. 1993. FASTUS: A finite-state processor for information extraction from real-world text. In *Proceeding of 13th International Joint Conference on Artificial Intelligence*, pages 1172–1178.

Argamon, S., I. Dagan, and Y. Krymolowski. 1999. A memory-based approach to learning shallow natural language patterns. *Journal of Experimental and Theoretical Artificial Intelligence, special issue on memory-based learning*, 10:1–22.

Baum, L. E., T. Peterie, G. Souled, and N. Weiss. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171.

Bengio, Y. 1999. Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162.

Bishop, C., 1995. *Neural Networks for Pattern Recognition*, chapter 6.4: Modelling Conditional Distributions, page 215. Oxford University Press.

Bourlard, H. and N. Morgan. 1989. Merging multilayer perceptrons and hidden Markov models: Some experiments in continuous speech recognition. Technical Report TR-89-033, International Computer Science Institute, July.

Bourlard, H. and C. J. Wellekens. 1989. Speech pattern discrimination and multilayer perceptrons. *Computer Speech and language*, 3:1–19.

Brants, T. 2000. TnT—a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000*, pages 224–231, Seattle, WA.

Braz, R., R. Girju, V. Punyakanok, D. Roth, and M. Sammons. 2005. An inference model for semantic entailment and question-answering. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI)*, pages 1043–1049.

Brill, E. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565.

Carlson, A., C. Cumby, J. Rosen, and D. Roth. 1999. The SNoW learning architecture. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science Department, May.

Carlson, A., C. Cumby, N. RizzoloJ. Rosen, and D. Roth. 2004. SNoW user manual, August. http://l2r.cs.uiuc.edu/~cogcomp/software/snow-userguide/.

Carreras, X. and L. Màrquez. 2004. Introduction to the conll-2004 shared task: Semantic role labeling. In *Proceedings of CoNLL-2004*, pages 89–97. Boston, MA, USA.

Carreras, X. and L. Màrquez. 2005. Introduction to the conll-2005 shared task: Semantic role labeling. In *Proceedings of Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 152–164. Ann Arbor, MI, USA.

Carreras, X., L. Màrquez, and J. Castro. 2005. Filtering-ranking perceptron learning for partial parsing. *Machine Learning, Special Issue on Learning in Speech and Language Technologies*, 59:1–31.

Charniak, E. 2001. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association of Computational Linguistics*, pages 116–123, Toulouse, France.

Chuang, J. and D. Roth. 2001. Gene recognition based on DAG shortest paths. *Bioinformatics*, 17(6):S56–S64, June.

Cohen, M., D. Rumelhart, N. Morgan, H. Franco, V. Abrash, and Y. Konig. 1992. Combining neural networks and hidden Markov models for continuous speech recognition. In *Proceedings of the DARPA Speech and Natural Language Workshop*.

Collins, M. 1999. *Head-driven Statistical Models for Natural Language Parsing.* Ph.D. thesis, Computer Science Department, University of Pennsylvenia, Philadelphia.

Collins, M. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8.

Cook, S. A. 1971. The complexity of theorem proving procedures. In *3rd annual ACM Symposium of the Theory of Computing*, pages 151–158.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein, 2001a. *Introduction to Algorithms*, chapter 29: Linear Programming, pages 770–821. MIT Press/McGraw-Hill, second edition.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein, 2001b. *Introduction to Algorithms*, chapter 24.2: Single-Source Shortest Paths in Directed Acyclic Graphs, pages 592–595. MIT Press/McGraw-Hill, second edition.

Cortes, C. and V. Vapnik. 1995. Support-vector networks. *Machine Learning*, 20:273–297.

Dagan, I., Y. Karov, and D. Roth. 1997. Mistake-driven learning in text categorization. In *Proceedings of EMNLP-97, The Second Conference on Empirical Methods in Natural Language Processing*, pages 55–63.

Dash Optimization. 2004. Xpress-MP. http://www.dashoptimization.com/products.html.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann, San Francisco, CA.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of Royal Statistical Society*, 39(1):1–38.

Dietterich, T. G. and G. Bakiri. 1995. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286.

Domingos, P. and M. Pazzani. 1997. Beyond independence: Conditions for the optimality of simple bayesian classifier. *Machine Learning*, 29:103–130.

Elkan, C. 1997. Bossting and naive bayesian learning. Technical Report CS97-557, Department of Computer Science, University of California, San Diego, September.

Freund, Y. and R. E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Garg, A. and D. Roth. 2001. Understanding probabilistic classifiers. In *Proceedings of the 12th European Conference on Machine Learning (ECML)*, pages 179–191.

Golding, A. R. and D. Roth. 1999. A Winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130.

Grefenstette, G. 1993. Evaluation techniques for automatic semantic extraction: comparing semantic and window based approaches. In *Proceedings of ACL'93 workshop on the Acquisition of Lexical Knowledge from Text*, pages 143–153.

Grishman, R. 1995. The NYU system for MUC-6 or where's syntax? In B. Sundheim, editor, *Proceedings of the Sixth Message Understanding Conference*. Morgan Kaufmann Publishers, pages 167–176.

Grove, A. and D. Roth. 2001. Linear concepts and hidden variables. *Machine Learning*, 42(1/2):123–141.

Gusfield, D. and L. Pitt. 1992. A bounded approximation for the minimum cost 2-SAT problems. *Algorithmica*, 8:103–117.

Hacioglu, K., S. Pradhan, W. Ward, J. H. Martin, and D. Jurafsky. 2004. Semantic role labeling by tagging syntactic chunks. In *Proc. of CoNLL-04*, pages 110–113.

Har-Peled, S., D. Roth, and D. Zimak. 2003. Constraint classification: A new approach to multiclass classification and ranking. In *Advances in Neural Information Processing Systems 15*, pages 785–792, Cambridge, MA. MIT Press.

Harris, Z. S. 1957. Co-occurrence and transformation in linguistic structure. *Language*, 33(3):283–340.

Jaeger, M. 2003. Probabilistic classifiers and the concepts they recognize. In *Proceedings of the 20th International Conference on Machine Learning (ICML-2003)*, pages 266–273, Washington, D.C.

Kakade, S., Y. Teh, and S. Roweis. 2002. An alternate objective funciton for markovian fields. In *Proceedings of the 19th Internal Conference on Machine Learning*, pages 275–282.

Kingsbury, P. and M. Palmer. 2002. From Treebank to PropBank. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC-2002)*, Spain.

Kleinberg, J. and E. Tardos. 1999. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. In *Proceedings of th 40th IEEE Symposium on Foundations of Computer Science*, pages 14–23.

Lafferty, J., A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, Williamstown, MA.

Li, S. 2001. *Markov Random Field Modeling in Image Analysis*. Springer-Verlag.

Li, X., P. Morie, and D. Roth. 2005. Semantic integration in text: From ambiguous names to identifiable entities. *AI Magazine. Special Issue on Semantic Integration*, pages 45–68.

Li, X. and D. Roth. 2001. Exploring evidence for shallow parsing. In *Proceedings of CoNLL-2001*, pages 107–110, Toulouse, France. Association of Computational Linguistics.

Li, Y., H. Zaragoza, R. Herbrich, J. Shawe-Taylor, and J. S. Kandola. 2002. The perceptron algorithm with uneven margins. In *Proceedings of the International Conference on Machine Learning*, pages 379–386.

Littlestone, N. 1988. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318.

Mackworth, A. K. 1977. Consistency of networks of relations. *Artificial Intelligence*, 8:99–118.

Marcus, M. P., B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, June.

McCallum, A., D. Freitag, and F. Pereira. 2000. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of the 17th International Conference on Machine Learning*, pages 591–598, Stanford, CA.

Mitchell, T. 1997. *Machine Learning*. McGraw Hill, New York, NY.

Morgan, N. and H. Bourlard. 1990. Continuous speech recognition using multilayer perceptrons with hidden Markov models. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pages 413–416.

Morgan, N. and H. Bourlard. 1995. Continuous speech recognition. *IEEE Signal Processing Magazine*, 12(3):24–42.

Muñoz, M., V. Punyakanok, D. Roth, and D. Zimak. 1999. A learning approach to shallow parsing. In *Proceedings of 1999 Joint SIGDAT Conference on Empirical methods in NLP and Very Large Corpora*, pages 168–178, College Park, MD.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Punyakanok, V. and D. Roth. 2001. The use of classifiers in sequential inference. In *Advances in Neural Information Processing Systems 13*, pages 995–1001.

Punyakanok, V., D. Roth, and W. Yih. 2005. The necessity of syntactic parsing for semantic role labeling. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1117–1123.

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285.

Ramshaw, L. A. and M. P. Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third Annual Workshop on Very Large Corpora*, pages 82–94.

Ratnaparkhi, A. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142, Philadelphia, PA.

Richard, M. D. and R. P. Lippmann. 1991. Neural network classifiers estimate bayesian a posteriori probabilities. *Neural Computation*, 3(4):461–483, Winter.

Rosen, J. L. 1999. Scaling up context-sensitive text correction. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Rosenblatt, Frank. 1957. The perceptron: A perceiving and recognizing automaton. Report 85-460-1, Project PARA, Cornell Aeronautical Laboratory, Ithaca, New York, January.

Roth, D. 1998. Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of AAAI-98*, pages 806–813, Madison, Wisconsin.

Roth, D. 1999. Learning in natural language. In *Proceeding of International Joint Conference on Artificial Intelligence*, pages 898–904.

Roth, D., G. K. Kao, X. Li, R. Nagarajan, V. Punyakanok, N. Rizzolo, W-T. Yih, C. Ovesdotter, and L. Moran. 2001. Learning components for a question-answering system. In *Proceedings of The Tenth Text REtrieval Conference (TREC 2001)*, pages 539–548, Gaithesburg, Maryland.

Roth, D., M-H. Yang, and N. Ahuja. 2002. Learning to recognize objects. *Neural Computation*, 14(5):1071–1104.

Roth, D. and W. Yih. 2002. Probabilistic reasoning for entity & relation recognition. In *Proc. of COLING-2002*, pages 835–841.

Roth, D. and W. Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Proceedings of CoNLL-2004*, pages 1–8.

Roth, D. and W. Yih. 2005. Integer linear programming inference for conditional random fields. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 737–744.

Roth, D. and D. Zelenko. 1998. Part of speech tagging using a network of linear separators. In *Proceedings of COLING-ACL '98*, pages 1136–1142.

Schrijver, A. 1986. *Theory of Linear and Integer Programming*. John Wiley and Sons.

Sha, F. and F. Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL*.

Shen, L. and A. K. Joshi. 2003. A snow based supertagger with application to np chunking. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 505–512.

Taskar, Ben, Carlos Guestrin, and Daphne Koller. 2004. Max-margin markov networks. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA.

Tjong Kim Sang, E. F. 2000. Noun phrase representation by system combination. In *Proceedings of ANLP-NAACL 2000*, Seattle, WA.

Tjong Kim Sang, E. F. 2002. Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 155–158. Taipei, Taiwan.

Tjong Kim Sang, E. F. and S. Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the CoNLL-2000 and LLL-2000*, Lisbon, Portugal. Association of Computational Linguistics.

Tjong Kim Sang, E. F. and F. De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proc. of CoNLL-2003*, pages 142–147.

Tjong Kim Sang, E. F. and H. Déjean. 2001. Introduction to the CoNLL-2001 shared task: Clause identification. In *Proc. of the CoNLL-2001*, pages 53–57.

Tjong Kim Sang, E. F. and F. De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada.

Tjong Kim Sang, E. F. and J. Veenstra. 1999. Representing text chunks. In *Proceedings of EACL'99*. Association of Computational Linguistics.

Toutanova, K., D. Klein, and C. D. Manning. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL 03*.

Valiant, L. G. 1998. Projection learning. In *Proceedings of the Conference on Computational Learning Theory*, pages 287–293.

Viterbi, Andrew J. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–267, April.

Xue, N. and M. Palmer. 2004. Calibrating features for semantic role labeling. In *Proc. of the EMNLP-2004*, pages 88–94, Barcelona, Spain.

Zhang, T., F. Damerau, and D. Johnson. 2001. Text chunking using regularized winnow. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 539–546.

Zhang, T., F. Damerau, and D. Johnson. 2002. Text chunking based on a generalization of winnow. *Journal of Machine Learning Research*, 2:615–637.

Author's Biography

Vasin Punyakanok was born on October 29, 1975 in Bangkok, Thailand. He received his Bachelor of Engineering in Computer Engineering from King's Mongkut's Institute of Technology Ladkrabang in 1995 with First Class Honors and top rank in his class. Before deciding to pursue advanced study abroad, he worked as a systems engineer at Kernel computers and communications for 2 years. In 1997, he flew to Urbana, Illinois to attend the Department of Computer Science at the University of Illinois at Urbana-Champaign where he received his Master of Science and Doctor of Philosophy, both under the supervision of Professor Dan Roth. His research focuses on Machine Learning and Natural Language Processing where he has developed fundamental tools and has published several papers in highly selective conferences. After his graduation, he will continue his research as a postdoctoral researcher in the Cognitive Computation Group at the University of Illinois at Urbana-Champaign.